Atelier B 4
User manual

Document written by ClearSy.

This documentation is distributed under the Creative Commons - CC-By.

Every names of products quoted in this document are trademarks of their respective authors.

ClearSy System Engineering
Parc de la Duranne - 320 av. Archimède
Les Pléïades III Bat A
13857 AIX EN PROVENCE CEDEX 3
FRANCE

Tel : 33 (0)4 42 37 12 70
Fax : 33 (0)4 42 37 12 71
email : contact@clearsy.com

Contents

1 Introduction **6**

1.1 About this document 6

1.2 Objets handled by Atelier B 6

1.3 Atelier B user interfaces 7

1.4 Organisation of this document 8

1.5 Conventions 9

2 Installation **10**

2.1 Preparing the installation 10

2.1.1 Introduction 10

2.1.2 Resources required before installation 11

2.1.3 Memory Requirements 11

2.1.4 Disk Space Requirements 11

2.1.5 External tools 11

2.1.6 Using Atelier B on a multi-platform network 12

2.1.7 Creating the Atelier B manager account 13

2.1.8 Choosing the Installation Directory 13

2.1.9 Users of Previous Version 14

2.1.10 Installation on a Read-only Partition 14

2.2 Installing Files 15

2.2.1 Windows 15

2.2.2 Mac OS X 17

2.2.3 GNU/Linux and Solaris 18

2.3 Previous versions of Atelier B 20

3 Overview **21**

3.1 Interfaces 21

3.2 Main GUI 22
3.3 Editor 23
3.4 Command Mode User Interface 25

4 Quickstart guide **28**

4.1 Projects management 30
4.1.1 Display the list of projects 31
4.1.2 Creating a project in Atelier B 32
4.1.3 Deleting a Project In Atelier B 34
4.1.4 Opening and Closing a B-Project 36
4.2 Projects : Advanced usage 38
4.2.1 Users management 38
4.2.2 libraries management 39
4.2.3 Definition files directories Management 40
4.2.4 Archiving a project 41
4.2.5 Restoration 43
4.2.6 Reading Properties of a Project 45
4.3 Components Management 47
4.3.1 Adding a Component 47
4.3.2 Suppressing Components 50
4.3.3 Displaying the List of Components 50
4.3.4 Reading Information on the Components 53
4.3.5 Editing a Component 54
4.3.6 Restoring a Component 55

5 Applying the B-method **57**

5.1 Presentation 57
5.2 Managing workspaces 58

5.2.1 creating a Workspace . . . 58
5.2.2 Workspace Modifications . . . 60
5.2.3 Deleting a Workspace . . . 62
5.2.4 Opening and Closing a Workspace . . . 62
5.3 Syntax analysis and Typecheck . . . 63
5.4 Automatic refinement . . . 66
5.5 Generating Proof Obligations . . . 66
5.6 Displaying Proof Obligations . . . 69
5.7 Automatic Demonstration . . . 71
5.8 Interactive Demonstration . . . 73
5.9 Cancelling Demonstrations . . . 73
5.10 Checking the Translatable Language (B0) . . . 74
5.11 Project Checking . . . 77
5.12 Translating . . . 78
5.13 Applying a Tool to all the Components of a Project . . . 79
5.14 Updating a Project . . . 80
5.15 Tools interruption . . . 81
5.16 Dependencies Management . . . 81

6 Analysing B developments 83

6.1 Presentation . . . 83
6.2 Project status . . . 84
6.3 Component Status . . . 85
6.4 Dependency Graphs . . . 87
6.5 Operation call graph . . . 90
6.6 Cross References . . . 91
6.7 Extracting Metrics . . . 93

7 B projects Documentation **98**

7.1 Description 98

7.2 Displaying a B Source 98

8 Atelier B Parameters Customization **101**

8.1 Introduction 101

8.2 Presentation of the customization system 101

8.3 Creating a resource file 102

8.4 Display resource values and AtelierB version 103

8.5 Modifying the external tools configuration 104

8.6 Modifying Memory Allocation of the Logic Solver 105

8.7 Modifying Memory allocation of the K parser 106

8.8 Configuring the Printing Script 106

Appendix **108**

Limitations of Project Documentation Tools 108

Files created by the Atelier B 109

1 Introduction

1.1 About this document

This documents aims to introduce the different features provided by the Atelier B tool.

Atelier B is a set of tools allowing developping applications using the B method.

Atelier B helps the designer in the formalisation of its application:

- By automatically executing actions relevant to the B method on specifications and refinements
- By providing helper services not directly required by the B method, but necessary for industrial developments, such as management analysis and documentation of a project.

1.2 Objets handled by Atelier B

In this section, we are introducing the main objects handled by Atelier B.

Component: File that contains a source written in the B language. It is the basis of a B development. A component is a generic term that can correspond to:

- a B specification (abstract machine),
- a refinement of this specification,
- its implementation (last refinement level).

Writing B component is performed by using the text editor of the workstation.

Project: Set of files (components, external C, C++, HIA or ADA sources, makefiles) that are used or created during the development of an application using B, completed with additional information used by Atelier B (see BDP).

User: It is necessary to define a list of users that are allowed to access and modify the project. Note that all users have the same access rights, and that at least one user should be provided.

Project Database: (BDP) All the internal information required for the correct execution of Atelier B tools are stored in a directory called BDP. This directory also contains files created by the Atelier B documentation tools.

1.3 Atelier B user interfaces

In this document, we call **B Tools** the tools that are related to the application of the B method, the analysis, the adjustement and the documentation of software written with B. The B environment described in this manual allows two main interfaces for using the B tools:

- an **interactive mode,** that uses a graphical interface based on windows and buttons. In the next sections, this mode will be called the *Graphical User Interface*
- a **batch interface,** that uses a dedicated language called **Command Language**. In the next sections, this mode will be called *batch mode*.

1.4 Organisation of this document

This document is separated in three parts, that allows to gradually initiate the reader to the use of Atelier B and its different interfaces.

It has been written with the aim of both allowing new users to learn Atelier B, and both allow user of previous versions to be efficient with this new version of Atelier B.

- The first section describes the installation of Atelier B and its prerequisites depending on the different supported operating systems. It also provides an introduction to the two interfaces.
- The second sections aims to provide a quick-start guide, that describes the different features available through the Atelier B interfaces.
- The third section explores the practical use of Atelier B within a B development.

1.5 Conventions

Each feature provided by Atelier B is presented in the following way:

- A *Description* section describes the main characteristics of the command,
- A *Graphical user interface* section, that shows how the command can be used through the graphical user interface
- A *Command-line interface* section that describes the command through the Atelier B batch mode,
- An optional *parameters* section documents the ressources that can be used to modify the behaviour of the command.

The parameters are described as follows:

Example	**Comment**
ATB*POG*Generate_Obvious_PO	name of the ressource
Configured during Atelier B installation	default value
Generate obvious proof obligations or not	Ressource description

Within the graphical user interface description, all the labels of buttons are written in italic. For instance the *Help* button.

2 Installation

This document describes the procedures to follow to:

- Install Atelier B,
- Configure and parameter Atelier B.

We recommend that you read the entire document before starting the Atelier B installation procedure.

Some specific kind of installation must be performed by the system administrator, as some operations must be performed by the "super-user" (creating login accounts, etc.).

The first part contains all the information necessary for selecting the machine and the filesystem where Atelier B will be installed. It also contains explanation on how to use Atelier B on a network.

The second part provides a detailed description of the procedure to follow in order to read the Atelier B files supplied.

After performing the operations described above, Atelier B and its user accounts can be potentially configured. These operations are described in paragraph 2.1.7.

The section 8 contains all the information necessary for customizing Atelier B.

The last part of this section explains the procedure to follow so as to ease the port of B projects developed with a previous version of Atelier B.

2.1 Preparing the installation

2.1.1 Introduction

This section explains how to:

- Select the machine to install,
- Use Atelier B on a network,
- Create the Atelier B manager account,
- Select the filesystem where the Atelier B files will be installed.

Note that those steps are not required on all operation systems. For instance, under Windows OS, it is not useful to create a manager account.

2.1.2 Resources required before installation

To install and use Atelier B, you must have one of the following operating systems available:

- Linux (with glibc version 2 or greater),
- Solaris 2.6 (SunOS 5.6) or compatible,
- Microsoft Windows (2000 or greater),
- MacOS (versions 10.4 or greater).

2.1.3 Memory Requirements

It is recommended to have at least 512 Mo of RAM.

The necessary memory space is strongly dependent of the other applications running on your workstation and of the size of the developments made with Atelier B.

2.1.4 Disk Space Requirements

To install Atelier B, you need about 100 Mo of disk space.

The disk space occupied by a project developed with Atelier B depends on the size of the B source files. The disk space occupied by the files generated automatically by Atelier B equals about 25 times the disk space of all the B source files.

2.1.5 External tools

Atelier B produces files that can be worked on with the following tools:

Tool	Version	Type	Vendor
LaTeX	2E	Word processor	Free software
PDFLaTeX	3	Word processor	Free software
Word	7 or greater	Word processor	Microsoft

Moreover, for certain fonctionnalities, it couls be usefull to install some free external tools :

- Dot, furnished with GraphViz [1]
- A SSH client (OpenSSH, or Putty for Windows users[2], ...)

[1] Visit `http://www.graphviz.org` for more informations

[2] See `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`

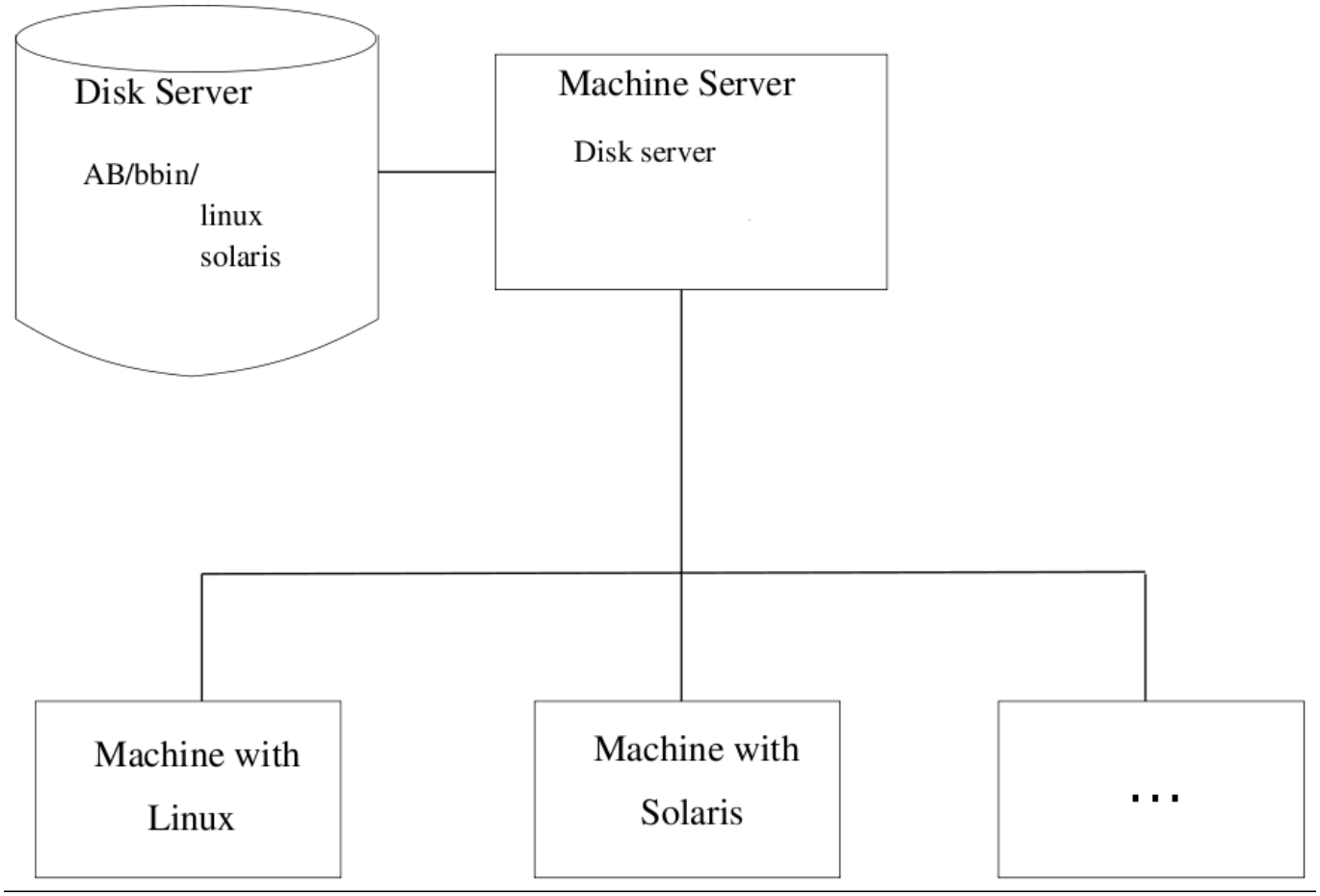


Figure 1: Example of a network system

You can use Atelier B without those tools but some functionalities will remain disabled.

2.1.6 Using Atelier B on a multi-platform network

You can install Atelier B on a network of machines that use a common file server. All the users of systems supported by Atelier B can share an installation directory. This manual contains instructions for installing Atelier B on a multi-platform network.

Figure 1 on page 12, shows an example of a network installation.

In this example, Atelier B is installed on a server machine.

Several machines, with various operating systems use Atelier B. The routing according to the type of system used is performed automatically by the Atelier B start-up scripts.

2.1.7 Creating the Atelier B manager account

This section concerns only UNIX-like operation systems in the case of a multiple-user configuration. The users can share projects and work on the same project at the same time.

The sharing of projects by several users means that the UNIX files must be shared between these users and therefore causes the usual problems of access rights between the users.

To avoid compromising system security, Atelier B uses a specific Unix user and group named *atelierb*.

Before installing Atelier B, you must therefore create an *atelierb* account and a group with the same name.

This user and this group must be defined on all the machines where Atelier B is likely to run.

The user *atelierb* has privileged rights on all the projects created via Atelier B. All intermediate files generated by Atelier B during a development will belong to the *atelierb* group.

The two Atelier B user interfaces are executable files with the "set-group-id" bit. Regardless of the user who executes them, Atelier B will have the rights of the *atelierb* group.

On the other hand, the user source files (B sources, proof rules files) will have the rights assigned by the users. They can therefore be protected with the standard UNIX access rights.

We recommend that you follow the procedures defined by the constructors for creating this account and this group:

- use `admintool` for SUN systems,
- use `sam` for HP-UX systems,
- use `adduser` or `linuxconf` for Linux systems.

We recommend using the directory selected in the sub-section below as the home directory of this account.

2.1.8 Choosing the Installation Directory

If you install Atelier B for several users, you must choose a directory where the users will have read and execute access. This directory must also be visible to all machines that are likely to run Atelier B.

To install the Atelier B files, you must have the write access to this directory.

We recommend that you install the Atelier B files in a directory named *atelierb* in the disk partition where you normally install applications.

To install Atelier B you will need around 100 Mbytes depending on your operating system.

2.1.9 Users of Previous Version

If you already have a previous version of Atelier B installed, you must install the new version in a different directory or in a different sub-directory of the previous directory.

Please refer to chapter 2.3 of this document. Note that if you tried a beta-version, you may will have to uninstall it before trying to install the current release.

2.1.10 Installation on a Read-only Partition

If for greater security you choose a disk partition that is "mounted" on other machines in read-only mode, you must follow a specific procedure when installing Atelier B.

By default, Atelier B uses a directory where the users must be able to write files. These files contain information on the projects created by the users.

This directory named `bdb` represents the *Atelier B data base*.

By default, this directory is located in the Atelier B installation directory. If the installation partition is read-only, you will have to use an Atelier B data base on another disk partition.

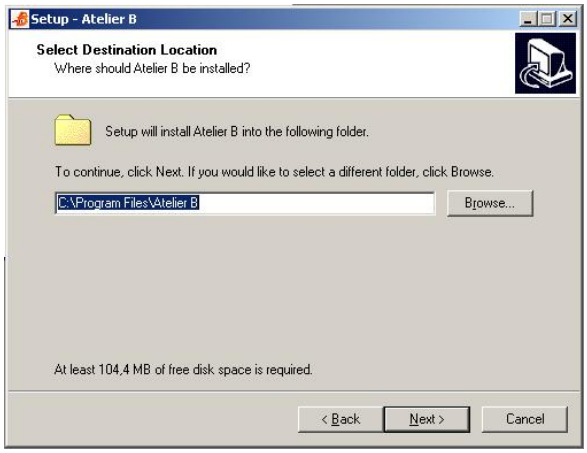
2.2 Installing Files

This section describes the Atelier B installation procedure and its options, for a workstation or a server.

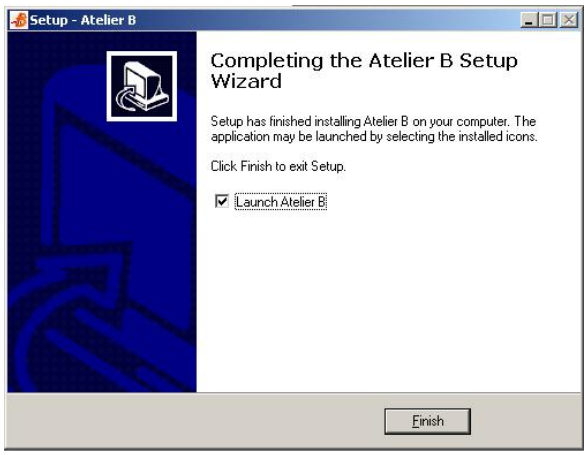
2.2.1 Windows

For this operating system, installation is fully automatic.

Just follow the assistant to simply install Atelier B. It will ask you for some preferences, for instance: the installation directory (see the figure below).



Finaly, if the installation is correct, it will display the following window:


Setup - Atelier B
Completing the Atelier B Setup Wizard
Setup has finished installing Atelier B on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
Launch Atelier B
Finish

2.2.2 Mac OS X

Under this operating system, Atelier B contains also an instaler with ".dmg" extension. Just follow the assistant to simply install Atelier B.

2.2.3 GNU/Linux and Solaris

An “atelierb” user could be necessary to install Atelir B under GNU/Linux or Solaris. This is not mandatory, as seen previously, but in multi-user mode it solves many problems concerning file access rights. Please see the previous section for more informations on “atelierb” specific user.

The installation under those operating system is done by a shell script After downloading Atelier B archive, you just need to uncompress it and to run the “install_atelierb” script.

```
user@host:~$ su atelierb
Password :
atelierb@host:/home/user$

[...]

atelierb@host:~$ tar xvzf atelierb-4.0-linux.tgz

atelierb@host:~$ cd atelierb-4.0/

atelierb@host:~/atelierb-4.0$ ./install_atelierb
```

Follow the instructions; below, the different steps proposed by the installation script:

```
------------------------------------------------------------------------


         ATELIERB Installation Procedure

Machine : host
Operating System : Linux 2.6.24-23-generic

Most of the questions that you will be asked have a default answer
enclosed in brackets, for instance:

         Do you understand [yes]?

To use the default answer, just type "return".
```

```
------------------------------------------------------------------------

Would you like to continue [yes]?

Installation procedure steps :

   1 - Extract installation files
   2 - Configure and parameter Atelier B products
   q - Quit installation procedure

Go to step number :  [1]?


Do you want to generate obvious proof obligations as a default [no]?
Enter the complete path to your text editor [/usr/local/bin/xemacs]?
Is Latex installed on your system [yes]?
Enter the directory containing Latex binaries [/usr/bin]?
Enter the name of the Latex viewer [xdvi]?
Enter the name of the Latex to PostScript generator [dvips]?
Is there a HTML viewer  installed on your system [yes]?
Enter the complete path to your HTML viewer [/usr/bin/firefox]?
Installation of Atelier B succeeded [OK]?
```

2.3 Previous versions of Atelier B

This section describes the procedure to follow so as to retrieve the projects developed with the previous version of Atelier B.

The procedure to follow is:

1. Archiving projects: With the previous version of Atelier B, you must archive the projects you wish to retrieve. To do this, use the "Archive project" function described in the "User Manual". You need only archive B source files and proof files.
2. Installing the new version of Atelier B: Follow the instructions of this manual to fully install the new version of Atelier B.
3. Restoring projects: With the new version of Atelier B, restore the projects which were archived previously using the "Restore Project" function as described in section 4.2.5.

3 Overview

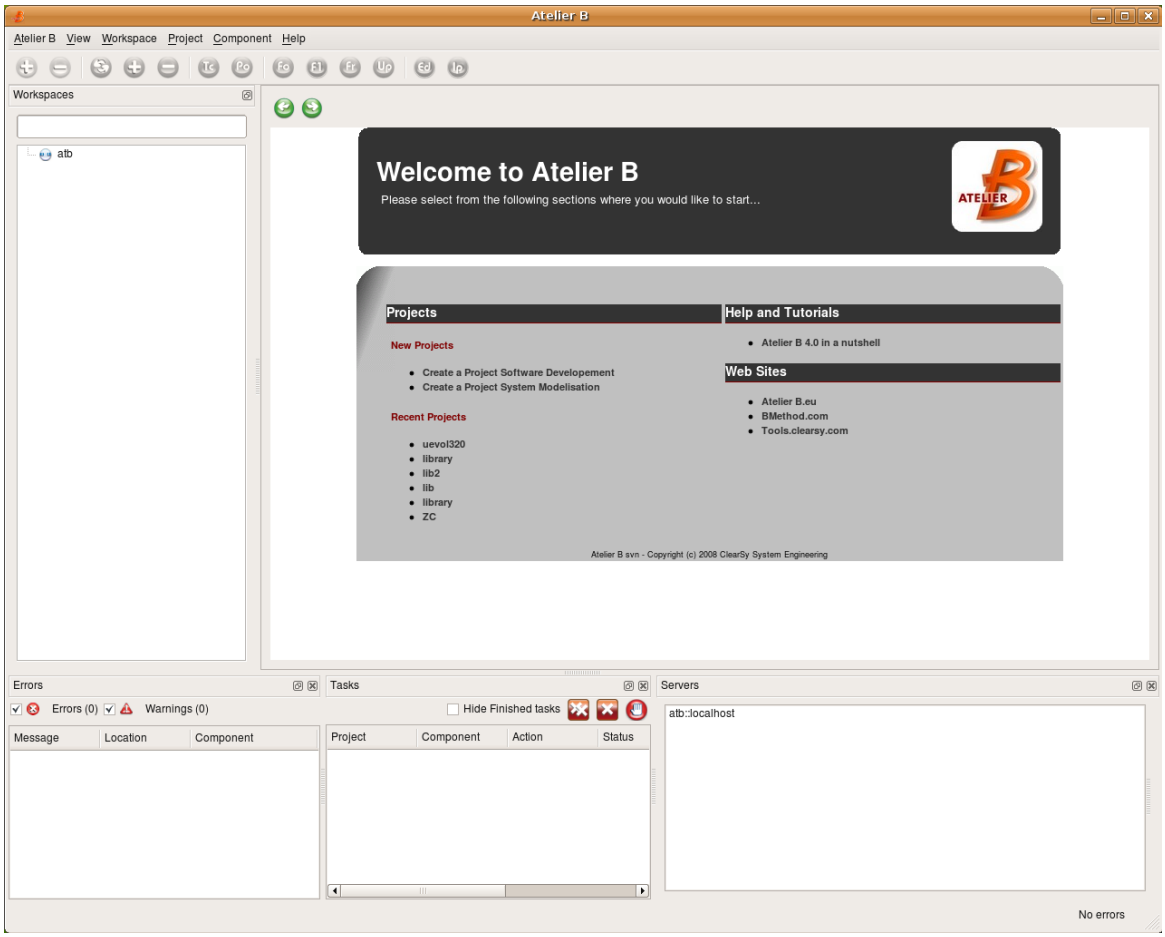
3.1 Interfaces

Atelier B user interface is split in two main parts : one dedicated to project management, the other allows editing B components using a specific-developped editor. Both have been developped in order to assist user during its development using the B method.

We will describe in details, in the following, this graphical user interface.

3.2 Main GUI

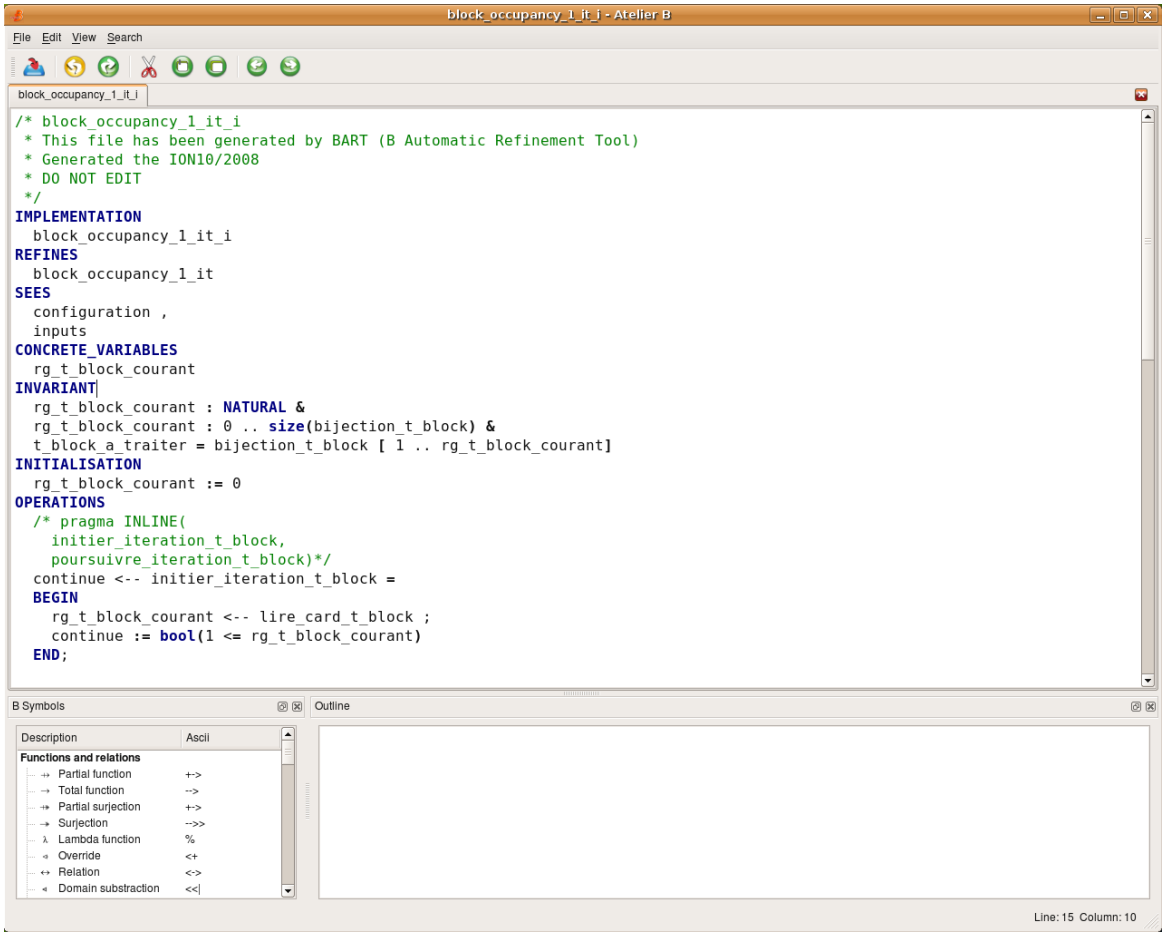
The main GUI looks like the following screenshot:



The main objective of this interface is to facilitate and optimize project management tasks.

3.3 Editor

Atelier B furnish a specific editor, specially customized for B component edition. It proposes fonctionnalities like: completion, error underlining, etc... It looks like this:



Its main functionalities are:

- Component editing,
- PMM file associated to B component editing,
- RMF file associated to B component editing,
- PatchProver and PatchRaffiner editing,
- Syntaxic coloration,
- Printing,
- Auto-completion and automatic indentation,

- Some syntax checking,
- Navigation between open files.

It also includes a presentation of language B symbols with facility to insert them in a component.

3.4 Command Mode User Interface

This user interface allows Atelier B to be used on a terminal as well as in semi-automatic mode with a command file.

This interface is a command interpreter (like a shell), with the mostly same features as the Atelier B graphical user interface.

To start up the command mode user interface, open a shell window and type the command :

```
$ ./startBB
```

Under Windows, a link exists in the Start menu : *"AtelierB Batch interface"*.

To quit the command mode user interface, type `quit` or `q`.

You can use this interface in two different ways:

1. interactively, or
2. with a command file.

Using a Command File

A command file can contain:

- Comments: lines starting with "`#`".
- Atelier B commands: long name or abbreviation.

Example of a command file:

```
#--------------------------
# This is a comment
#--------------------------
# list of projects
show_projects_list
# users of the LIBRARY project
spul LIBRARY
# contents of the current directory
\ls -l
# end of the command file
```

To execute a command file (only under UNIX-like OS), type one of the following commands:
`startBB -i=file_name`

or
`startBB < file_name`
or
```
startBB << END
list of commands...
END
```

Using the Interpreter

Some commands have default parameters. For project management commands, the interpreter stores the last project name entered. In the same way, the interpreter stores the last component name entered. To use this default parameter, simply type `<return>`.

Example:

```
bbatch 3> typecheck MM_1
....
bbatch 4> pogenerate
pogenerate MM_1 ? (yes=return)
...
```

Using Interactive Help

The `help` command displays the list of commands available. This list is displayed in the following format:

```
General commands :
(cd  ) change-directory
...
(v   ) version-print

Project level commands:
(add ) add-definitions-directory
(apl ) add-project-lib
...
(spl ) show-projects-lists

Machine level commands (available after open_project):
(aa  ) ada-all               (a   ) adatrans
(af  ) add-file              (ani ) animator
....
....
(s   ) status                (sg  ) status-global
(t   ) typecheck             (u   ) unprove
```

The commands are presented in the following order:

1. General commands,

2. Project management commands,
3. Commands that apply to components: A project must be opened to be able to use these commands.

Command abbreviations are indicated within parenthesis () before the command name.

You can also obtain help on a specific command by typing the command: `help command_name`, or `h command_name`.

For example:

```
bbatch 9> help help
help [command]    get help on commands
```

4 Quickstart guide

The next chapters of this document describe every functionalities supplied by the Atelier B.

This chapter describes more particularly the steps to follow in order to use the functions of the Atelier B.

In summary, to start a development with the Atelier B, you will have to :

1. Create a project (see sections #4.1.2),
2. Open this projet (see section #4.1.4),
3. Add components to this project (see section #4.3.1).

Or simpler, if you have a project archive, you could restore it (see section #4.2.5) and begin to work with the newly restored project.

Once those steps accomplished, you can begin to apply B Method on your components ; the Atelier B allows you to :

1. Perform the syntax analysis and the Type Check (see section #5.3),
2. Generate the Proof Obligations (see section #5.5),
3. Automatically demonstrate some of these Proof Obligations (see section #5.7),
4. Print the Proof Obligations (see section #5.6),
5. Use the interactive prover to demonstrate remaining proof obligations (see section #5.8).

After having created the implementations of your project, you will be abled to :

1. Control the language of these implementations (see section #5.10),
2. Translate the project into machine language by using a translator (see section #5.12).

During those development steps, you can use the analysis functionalities of the Atelier B, in order to :

1. Print the progress state of the project or a component in particular (see sections #6.2 and #6.3),
2. Show dependence graphs between components (see section #6.4).

You could similarly use functionalities in order to produce automatically documentations in various formats (Plain Text, LaTeX, PDF, or RTF ; see section #7.2 for more information about these features).

When your B Projects are reaching a significant size, you can :

1. Archive them in order to do backups (see section #4.2.4),
2. Split your large projects into several little projects by using the concept of libraries (see section #4.2.2).

4.1 Projects management

A project managed by the Atelier B is characterized by :

- Its name,
- A Project Data Base where all the files created by Atelier B are stored (PDB),
- A directory where the translations of components to machine languages are stored,
- A set of B source files; these files can be located in different directories.

The creation of these directories and B source files is entirely up to the user if using Command-line tool. The graphical user interface creates the directories if necessary.

The projects managed by Atelier B are multi-user projects. Many users can work on the same project simultaneously.

Atelier B uses in its ideal installation, the rights of the UNIX group *atelierb* in order to solve UNIX files access permitions between these users (see paragraph #2.1.7).

It is however possible to use Atelier B in "mono-user" mode, you would although lose in end-use flexibility.

A B project, managed by Atelier B, can also use other projects regarded as libraries. Libraries may be used to split industrial size projects into some smaller projects.

Atelier B also offers functions to archive projects. These functions are used to back-up or copy projects.

For project archiving and portability reasons, we encourage Atelier B users to adopt a project layout that obeys the following rules:

- All users of the same project should be in the same UNIX group (the *atelierb* group).
- The project directories should ideally have a common root. If `project_dir` is this directory, the PDB and the translation directory must be sub-directories of `project_dir`. For example: `project_dir/pdb` and `project_dir/lang`.
- The B source files must be located in sub-directories of `project_dir`. You can for example define a sub-directory for each software element, especially if the users who are responsible for them are different.
- The directories containing definition files (see section #4.2.3) must also be sub-directories of `project_dir`.

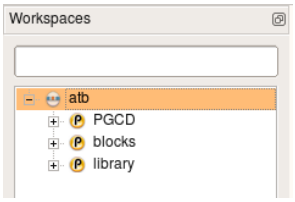
4.1.1 Display the list of projects

Description

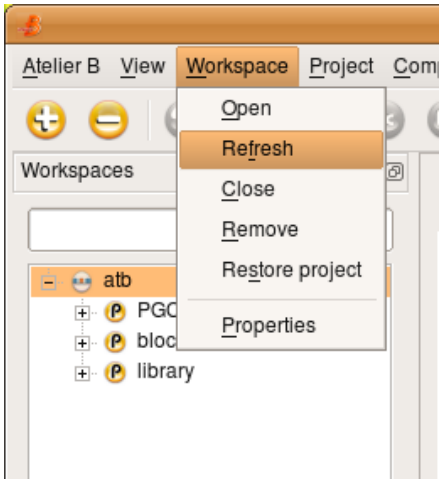
This function is used to display the list of projects accessible to the Atelier B user.

Graphical user interface

Once the workspace is opened, the projects list is directly visible into the main window of Atelier B, as seen on this screenshot :



After some modifications, it may be useful to update the list ; this could be done by clicking onto the "*Workspace -> refresh*" menu.



Command-line interface

The user interface has already been started up.
To view the list of projects, type the following command:

```
show_projects_list
or
spl
```

The list is displayed as shown below :

```
Printing Projects list ...

     project1
     project2

End of Projects list
```

4.1.2 Creating a project in Atelier B

Description

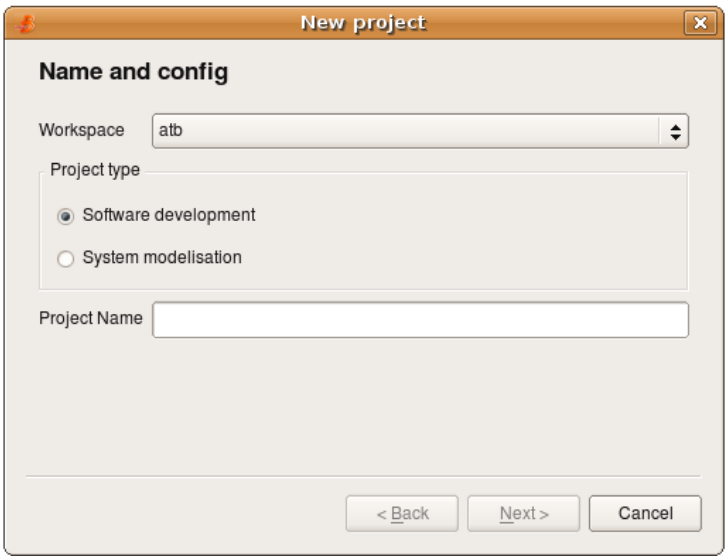
The *Attach project* function allows all users to declare a new project in Atelier B. The information required to declare the project is:

- the name of the new project and its owner,
- the Project Database directory (PDB) used by Atelier B to store internal files,
- the translation directory used by Atelier B to store translated files.
- The project type (non required) : SOFTWARE for B-Software projects, SYSTEM for Event-B projects. By default, the newly created project is considered as software.

To reference a given project, only the PDB and translation directories are required by Atelier B (refer to the directions below). The creation of these two directories is up to the user. The project components can be "scattered through" the file system, their access path and names are stored in a file located in the PDB and called `project_name.db`.

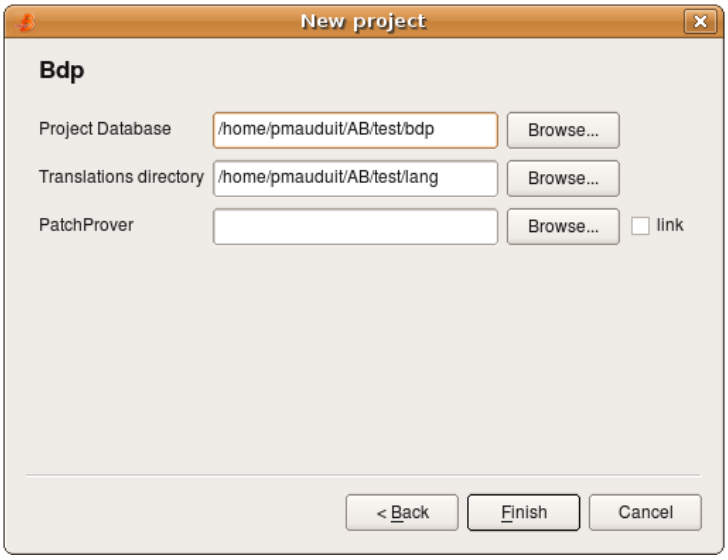
Graphical user interface

In order to create a project from the GUI, you have to click on the "*AtelierB -> New Project*" menu, or use either the shortcut "Ctrl+P", either a right click on the given workspace, then the "*New -> Project*" menu. The project creation wizard will then appear:



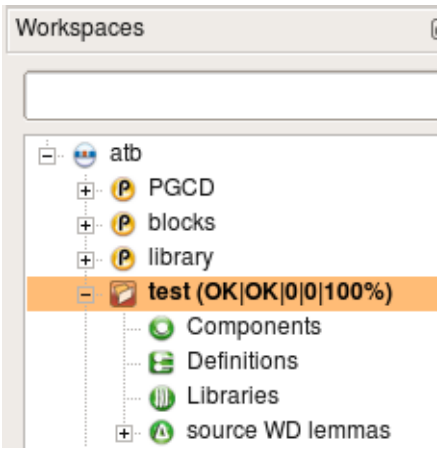
You will have to choose between B-Software and Event-B.

After having typed a project name, the interface will choose directories suitable for the project location, but it will be customizable on the second page of the wizard :



You can use the *"Browse"* buttons to facilitate the choice of these directories.

Once the configuration is done and the wizard closed, the newly created project should appear into the workspace view, and be automatically opened, as shown in the screenshot below :



Command-line interface

To create a project:

- Choose and create a directory for the PDB,
- Choose and create a directory for the translations,
- Launch the command-line interface,
- Create the B project by typing the following command :

```
create_project nom_projet chemin_bdp chemin_lang SOFTWARE
or
crp nom_projet chemin_bdp chemin_lang SOFTWARE
```

- Check that the project has been correctly created by typing:

```
show_projects_list
or
spl
```

Remarks: You are not forced to type in the full path ; you can give a relative path according to the current directory from where you launched the Atelier B. The previous command allows to create a B-Software project. If you want to create an Event-B project, just replace "SOFTWARE" by "SYSTEM".

4.1.3 Deleting a Project In Atelier B

Description

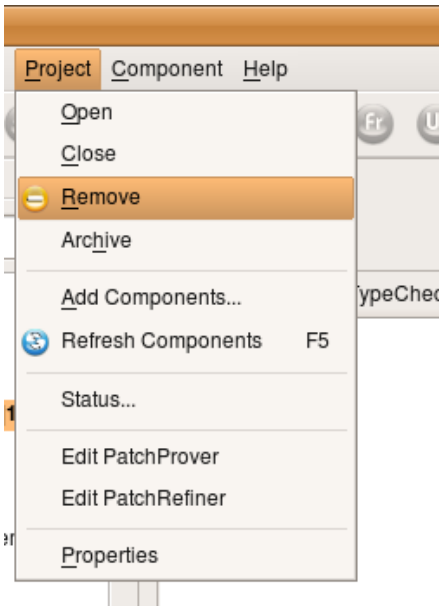
The *Detach project* function deletes an existing project. The intermediate files produced by Atelier B in the project data base will then be deleted. These informations are not deleted :

- B source files,
- user rules files (`*.pmm`),
- the project documentation that is automatically generated,
- the translations,
- the project data base directory,
- the translations directory.

To completely clear the project you should delete manually those files and directories after deleting the project using Atelier B.

Graphical user interface

This action is available from the *"Project -> remove"* menu, or from a right click on the workspace view :



The files created by Atelier B are destroyed, although some information are kept.

Command-line interface

The user interface has already been started up.
To delete a project named `proj`, type the following command:
`remove_project proj`
or
`rp proj`

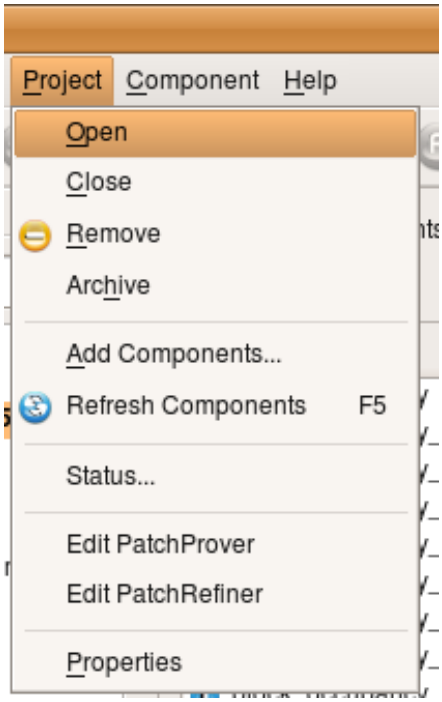
4.1.4 Opening and Closing a B-Project

Description

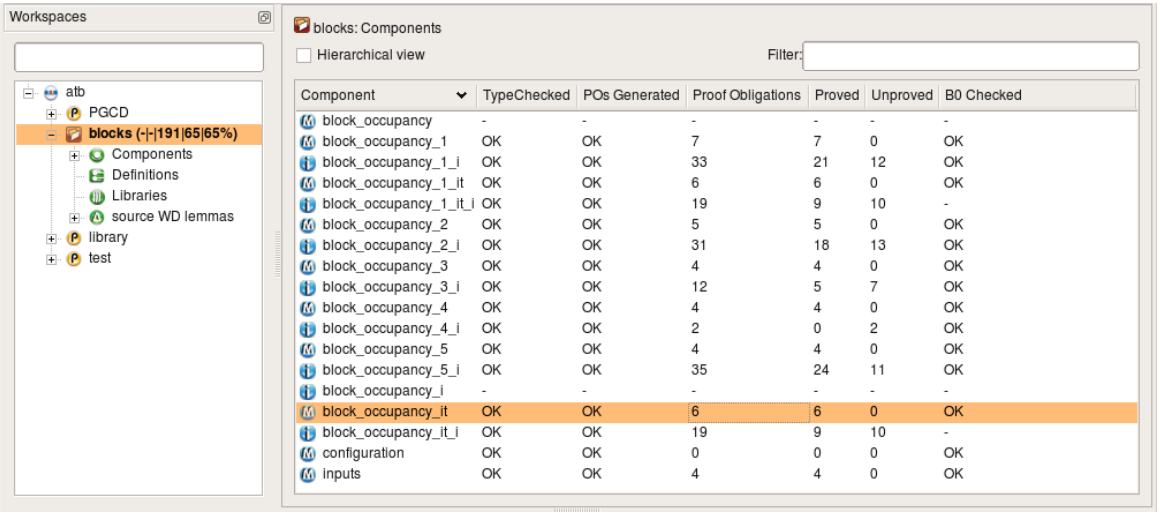
The function *"Open project"* grants access to the components of a project. Reciprocally, the function *"Close Project"* closes the previously opened project.

Graphical user interface

To open a project into the interface, you have to, click on the *"Project -> Open"* menu, or to right-click on the given workspace on the wanted project, then click on *"Open"*.



Once the project is opened, the components' list appears :



This view is customizable, and it is possible to sort this list according to a given column by clicking on its header.

This version allows you to open several projects simultaneously.

Reciprocally, the closing of a project is triggered by using the *"Project -> Close"* menu of the interface. This leads to empty the components' listing.

Command-line interface)

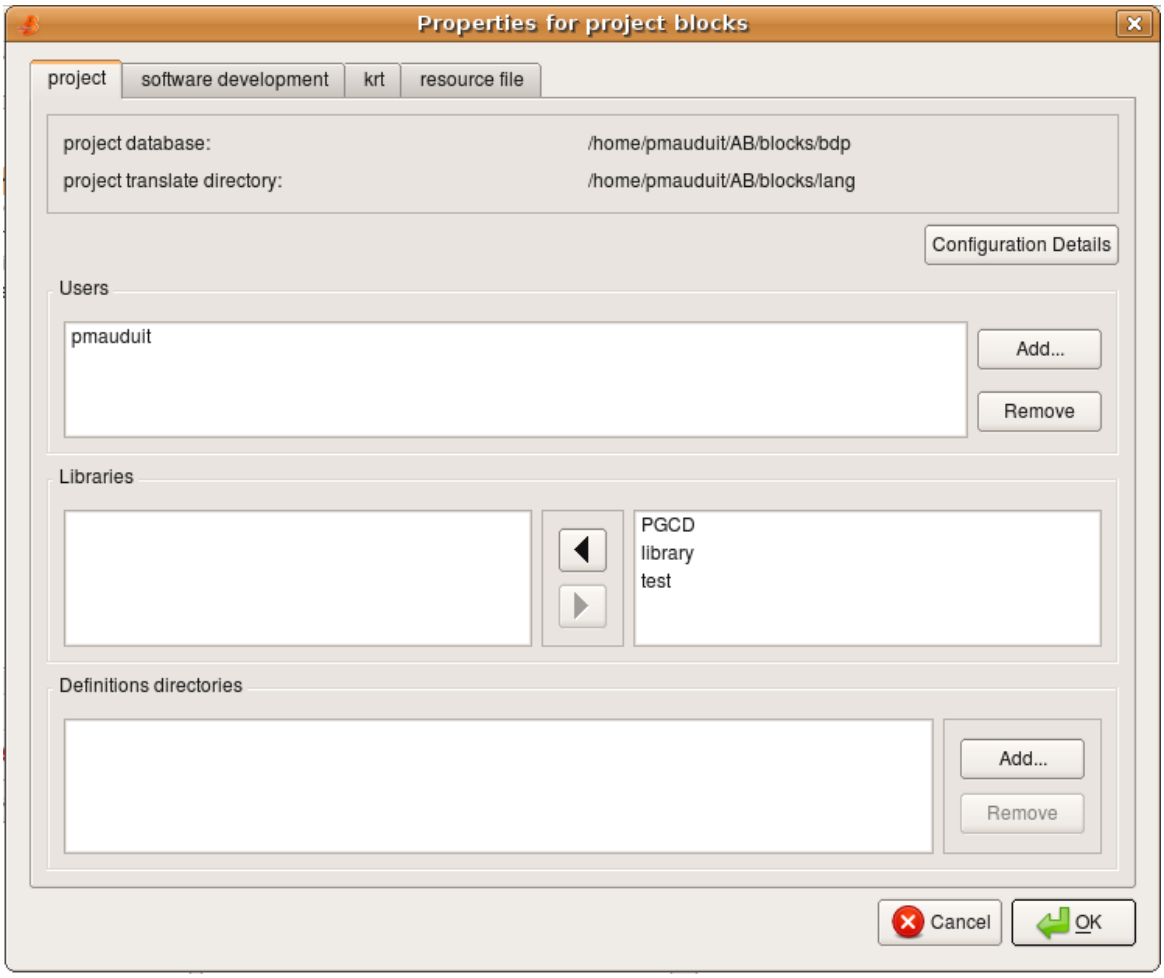
The user interface is already started up.
To open project `proj_name`, type the following command:

```
open_project proj_name
```
or
```
op proj_name
```

The user interface is already started up.
To close a project, type the following command:
```
close_project
```
or
```
clp
```

4.2 Projects : Advanced usage

This section is about the advanced management of projects. The main functionalities described below are available using the *"Project -> Properties"*, which gives access to the following interface :



The command-line equivalents will be given afterwards.

4.2.1 Users management

Description

In order to grant access to the objects managed by Atelier B, it is possible to add and/or delete users ; in addition, every users declared on a project have the same read/write rights on it. It is however essential to let at least one user in the list.

Graphical user interface

Simply click on the users you wish to suppress, then click on the *remove* button.

Click on the *Add ...* button to display a dialog box allowing you to type in a user name. If you want to give an access to every users, you can specify a wildcard using the character '*'.

4.2.2 libraries management

Description

When developping on significant-sized projects, it is fundamental to be able to :

- Use libraries of predefined components,
- Structure a prominent project in several sub-projects

If they wish, the users can bind their projects to other projects managed by the Atelier B. For this, they can use the *Add library* function.

Every project available can become a library to the current project.

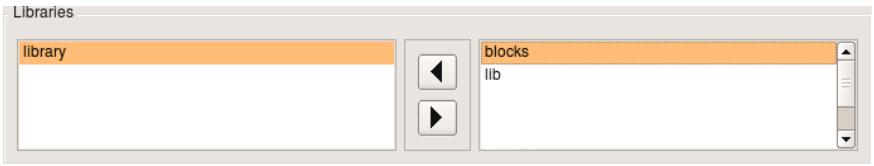
When a library is bound to a project, the user of the project can write links (SEES, IMPORTS, ...) to components provided by this library.

The *Add library* function checks that the library to add is not already declared within the project.

If a component is defined into several libraries, then the component which would be evaluated would be the one which is defined into the first imported library. In case of doubts on the handled components, it could be useful to request a printing of the project's dependence graph.

Graphical user interface

As displayed on the previous screenshot, the potential libraries list is displayed into the frame *Libraries*, into the box on the right. Use the mouse to select the project to be added as a library then click on the left-pointed arrow button in order to add it to the current project.



Reciprocally, select the library you want to exclude from the project, then click on the right-pointed arrow button, in order to exclude a library from the project.

Command-line interface

With the user interface running. To add the `lib_name` library to the `proj` project, type the following command:

```
add_project_lib proj lib_name
```
or
```
apl proj lib_name
```

To display the list of libraries of project `proj_name`, type the following command:
```
show_project_libs_list proj_name
```
or
```
spll proj_name
```

The list is displayed as shown below:

```
Printing Project proj_name libs list ...

        lib1
        lib2

End of Project proj_name libraries list
```

To remove the `lib_name` library from the `proj_name` project, type the following command:
```
remove_project_lib proj_name lib_name
```
or
```
rpl proj_name lib_name
```

4.2.3 Definition files directories Management

Description

Definition files are a means to share common definitions for several components.

Their description are given in chapter *2.3 The DEFINITIONS clause* of the document *Reference Manual for B language* .

This section describes the procedures to follow so as to add a new directory to a B project, which can contain definition files used by the components of this project.

Graphical user interface

Still onto the project properties page (see the previous screenshot), into the *"Definition directories"* frame, click on the *Add* button to display a dialog box which allow you to select a directory.

On the contrary, to delete a definition file directory, click on the one you want to suppress then click on the *Remove* button.

Command-line interface

The user interface has already been started up.

To add the `dir` directory to the `proj` project, you must type the following command :
`add_definitions_directory proj dir`
or
`add proj dir`
Beware, you must indicate the absolute path of the directory.

On the contrary, To delete the `dir` directory from the project name `proj_name`, you must type the following command :
`remove_definitions_directory proj_name dir`
or
`rdd proj_name dir`.

4.2.4 Archiving a project

Description

This function allows you to archive every files of a given project. The created archive is a zlib-compressed file equivalent to a classical tar file.

This function can be used to:

- back-up a project,
- make a copy of a project (e.g., to transfer it to another machine).

There are three options for archiving:

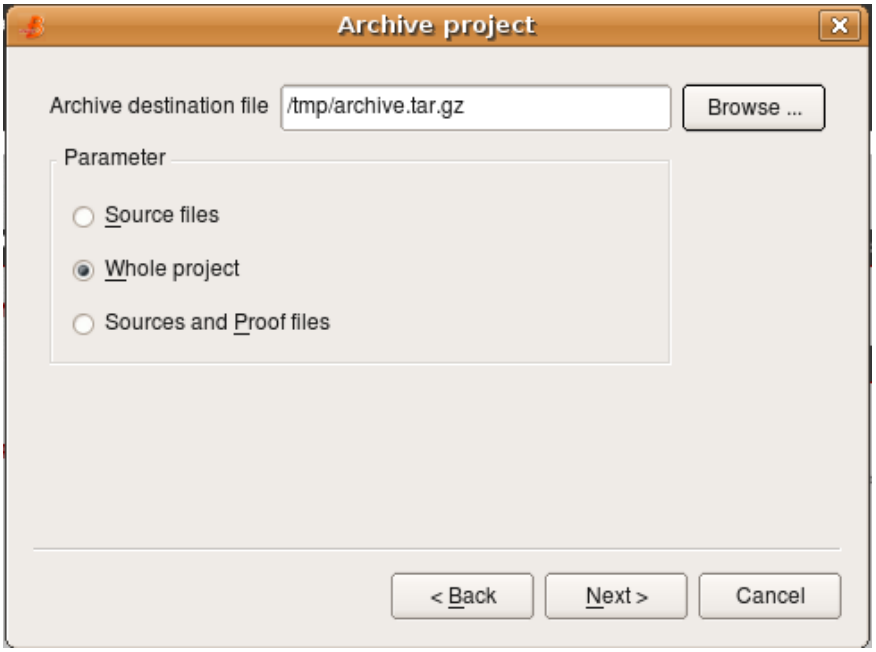
- Save all the B source files (`.mch`, `.ref` and `.imp` and definition files).

- Save all the B source files and the proof files.
  With this option, Atelier B also saves the files in the project data base used by the proof tools:
  - `.po` files: proof obligations,
  - `.pmi` files: demonstrations,
  - `.pmm` files: user rules,
  - `PatchProver` file: user tactics,
  - `AtelierB` file: project resource file.
- Save the entire project:
  - B source files,
  - files present in the project data base directory,
  - files present in the translation directory.

When "all" the project is archived, all data is stored. As a result, when the project is restored (refer to sub-section #4.2.5) the user will retrieve it in the same state; and will not have for instance to type check again what was already type checked.

Graphical user interface

In order to archive a project with the GUI, it is necessaryn after having opened the workspace containing the project to archive, to click on the given project, and to select into the menu : *Project -> Archive*. A wizard is displayed then :



Select a destination file, and the different parameters needed, then click on the *Next* button.

Command-line interface

The user interface is already started up.
To archive a project, use the `archive` or `arc` command.

This command has three parameters:

- The project name,
- The path to the archive file,
- The type of archival:

  0 B source files archive,
  1 entire project archive,
  2 B source and proof files archive.

Example:
To archive the entire project `proj_name`, type the following command:
`archive proj_name /home/project/tarPROJ.arc 1`

The archive created is named `/home/project/tarPROJ.arc`.

4.2.5 Restoration

Description

This function is used to restore a project managed by Atelier B from the data stored in an archive created using the function described in the previous sub-section.

Three restore options are available:

- restore B source files,
- restore B source files and proof files,
- restore the entire project.

A restoring always creates a new project. When a project is restored the following data from the archived project are lost:

- the list of project libraries,
- the name of the project manager; the user who is restoring the project automatically becomes the manager of the created project,

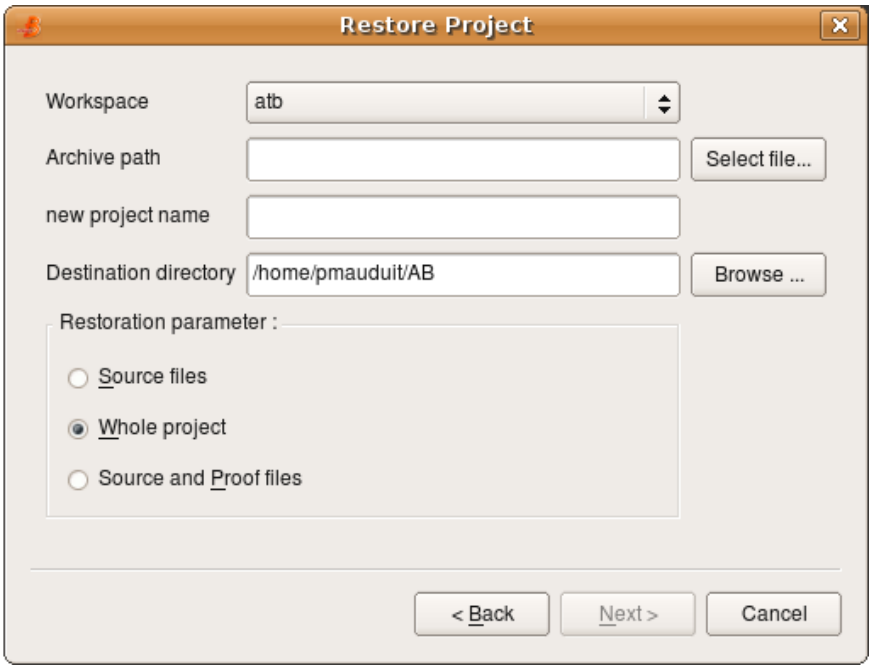
- the list of users authorized to access the project.

Remark: You can restore the B source files and the proof files of a project archived with a previous version of Atelier B.

You cannot restore a project using the *"sources and proof files"* option if your archive has been generated with the *"only B sources"* option. If so, the Atelier B will indicate you an error.

Graphical user interface

A wizard has been implemented to lead you in the restoration task :



First choose a workspace in which you are willing to restore your project, then choose a path to the archive file ; give then a name to your newly restored project, and a path in which to restore the files. Finally, select the restoration parameter, and click on the *"Next"* button.

If the restoration process has successfully finished, the new project must appear in the workspace. The components and the definition directories are automatically attached to the project.

Command-line interface

The user interface is already started up.
To restore a project, use the `restore` or `res` command.

This command requires four parameters:

- the archive file access path,

- the type of restore:

  0 restore B source files,

  1 restore the entire project,

  2 restore B source files and proof files.

- the name of the restored project: you must give the name of a new project.
- the project base path, if you do not wish the directories to be created in the current directory.

Example:
To restore and create the `COPY` project, type the following command:
`restore /home/project/tarPROJ.arc 1 COPY /home/COPY`

A new project called `COPY` is created in the `/home/COPY` directory.

4.2.6 Reading Properties of a Project

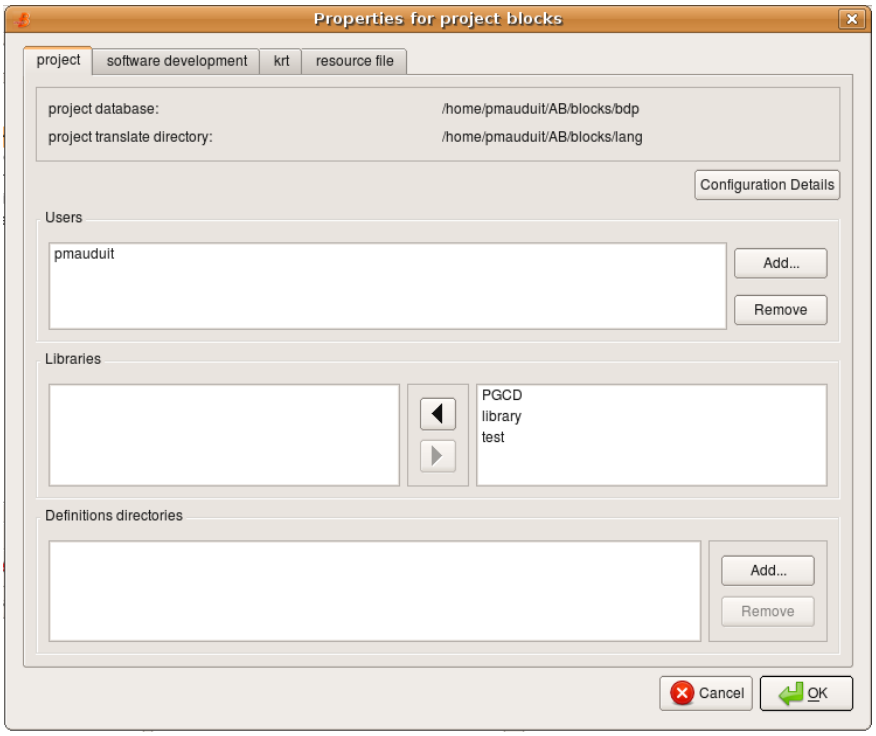
Description

This function displays the following properties of a project:

- the access path to the project database,
- the access path to the translations directory,
- the name of the project manager,
- the list of project libraries.

Graphical user interface

The GUI is already launched, and a workspace is opened. To obtain information on a project, you have to display the *Project Properties* page :



Command-line interface

The user interface is already started up.
To view the properties of project `proj_name`, type the following command:
`project_infos proj_name`
or
`ip proj_name`

The properties are displayed as follows:

```
Name    : LIFT
Database path : /home/projects/pdb
Translations path : /home/projects/transl
Manager : user
Libraries : LIBRARY
```

4.3 Components Management

4.3.1 Adding a Component

Description

A B project is made up of a list of B components located in text files.

These components are either directly linked to the project, or accessible through libraries.

The B source files are text files containing one or several components.

Once a source file has been attached to a project, the user analyses which components are contained in the file and permit the management of these components.

Atelier B only accepts as B source files those whose name ends in one of the four following suffixes : `.mch`, `.ref`, `.imp`, `.mod`. The contents of each type of file is as follows :

- A file named `Ident.mch` must contain one and only one component : an abstract machine named `Ident`.
- A file named `Ident.ref` must contain one and only one component : a refinement named `Ident`.
- A file named `Ident.imp` must contain one and only one component : an implementation named `Ident`.
- A file named `Ident.mod` must contain a component named `Ident`. It can also contain the refinements of this component, as well as modules (an abstract machine and all its refinements) imported by the implantation of the component. Finally, for each module present in the file, the modules imported by this module can also be present. A suffix file `.mod` permits the storage in a single file of a B module, or a sub-part of the project made up of a module and the modules imported by this module.

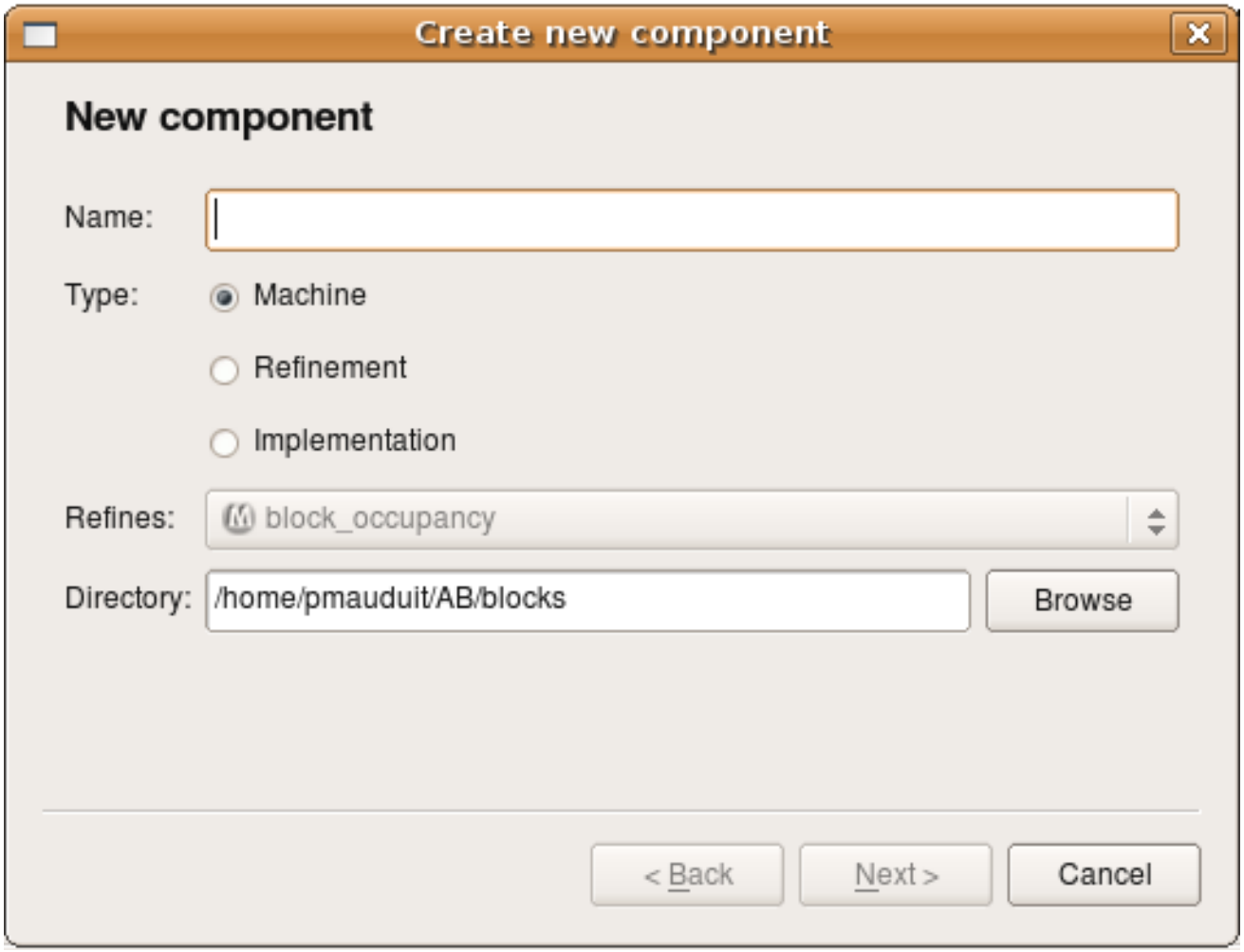
If the contents of the files do not respect these rules, the file will still be attached to the project. An error message is then displayed in the start up window. No other operation will be possible on the component before its correction.

Graphical user interface

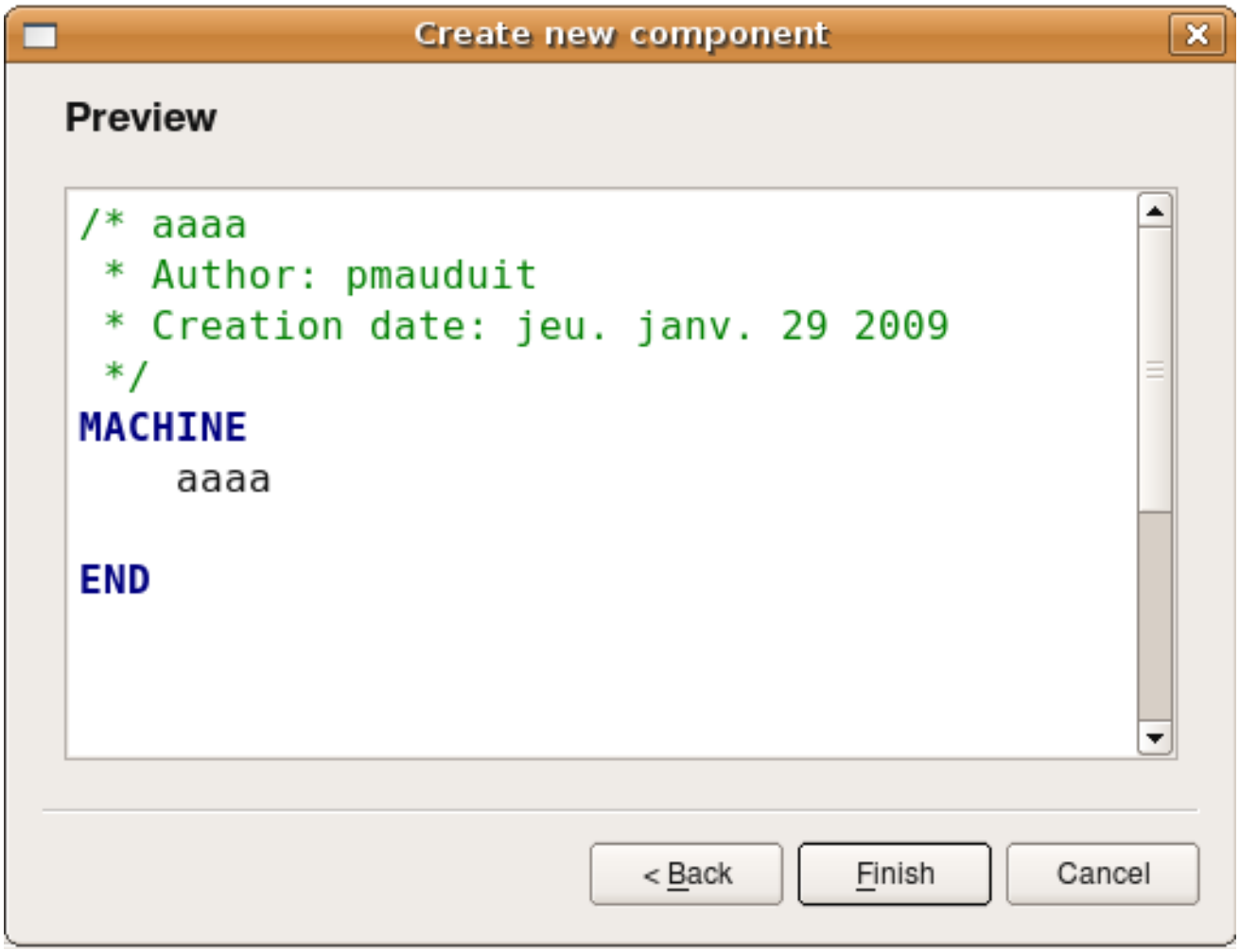
The GUI is launched, and a project is opened. In order to add components from existing files, you can use the *"Project -> Add Components ..."* menu. This will lead you to a file selection dialog. Note that the file selection can be multiple if you have to add more than one file. Once the desired files are selected, confirm by clicking on the *"Open"* button.

An other solution will be to create a new component, if it does not exist. To do so, select in the contextual menu *"New -> Components ..."*, or click in the blue "+" icon onto the toolbar. It will then

display the following wizard :



Once the name of the new component, the component type, the potential raffined component and the destination path are entered, click on the *"Next"* button. The wizard will then show you a overview :



Command-line interface

The user interface is started up, you already have opened a project.
To add a component to the current project, perform the following operations:

1. Go to the directory where the B source file is located, using the command
   `cd` or
   `change_directory`
2. Display the list of B sources present in this directory, using the command `lsb` or `list_sources_b`.
   This command displays the list of files with `.mch`, `.ref`, `.imp` or `.mod` extensions present in the current directory.
3. Add a component using command:
   `af file_name` or `add_file file_name`

Example:
`af AA.mch`

4.3.2 Suppressing Components

Description

This function permits to suppress one or several components from the current project.

B source files themselves are not deleted, they are just removed from the list of components of the project.

When a component is suppressed, all the information related to this component are deleted from the PDB, except the following ones:

- B source files,
- user rule files (`*.pmm`),
- automatically generated project documentation,
- translations.

Graphical user interface

The GUI is launched, and a project is opened. Select the components you wish to exclude from the project, and select the *"Component -> remove ..."* menu. (Or you can also click on the blue "-" icon into the toolbar).

Command-line interface

The user interface is started up, you already have opened a project.
To suppress a component from a project, type the following command:
`remove_component comp_name`
or
`rc comp_name`
If `comp_name` corresponds to a component part of a `.mod` file then all the components included in the `.mod` file will be deleted.

4.3.3 Displaying the List of Components

Description

This function displays the list of components of a project, grouped by B source files names.

The components are displayed in alphabetical order, according to the names of B source files.

If the components `BB`, `BB_1` and `AA` are present in the same B source file (file `.mod`) then the components will be displayed in the following way:

```
BB  AA
    BB
    BB_1
```

On large scale projects, the number of components is often very important. This function offers several filters permitting to reduce the number of components and to search components by their names.

The available filters are:

- Filter according to the component manager user.
- Filter by type of component: machine, refinement, implementation.
- Filter by component name.

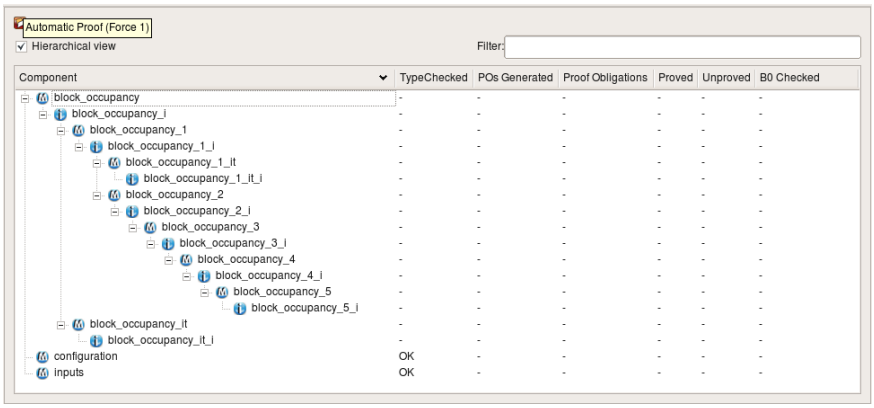
Graphical user interface

The GUI is already launched, and a project is opened. The list of components is printed permanently.

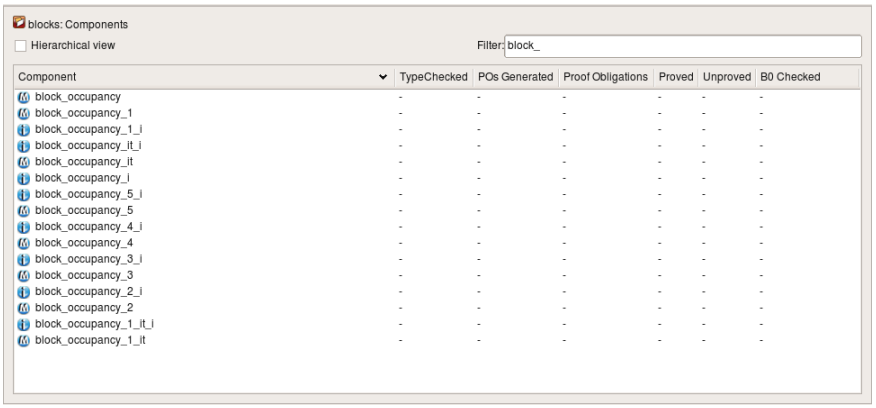
The filters available inside the interface are different comparing to the one of the Command-line interface ; it is possible to :

- Filter on a component name (by giving a search filter into the textbox located on top of the component view),
- Activate the hierarchical view, which will sort the components following a refinement arborescence,
- Filter on a specific column.

See below an example of hierarchical display :



On the next screenshot is displayed a usage of the *"Filter by name"* functionality :



command-line interface

The user interface is started up, you already have opened a project.
To display the list of project components, use the following command:
`show_machine_list` or `sml`.

This function accepts five optional parameters:

own If this parameter equals `1` only the components that you are manager of are displayed.

mch If this parameter equals `0` the machines are not displayed.

ref If this parameter equals `0` the refinements are not displayed.

imp If this parameter equals `0` the implementations are not displayed.

name This parameter allows to filter the list by component names. Use the `*` character. For example, to display all components with a name starting with a S letter, specify the `S*` value.

Examples:

To obtain the list of the components that you manage:

```
bbatch 2> sml 1
Printing Own machines list ...

      B_Site_central
      Card Card
           Card_imp
      Keyboard_code
      Keyboard_code_ref

End of machines list
```

To list machines only:

```
bbatch 2> sml 0 1 0 0
Printing  machines list ...

      B_Site_central
      Card Card
      Keyboard_code

End of machines list
```

To list only machines with a name starting with `K` and ending with `code`:

```
bbatch 2> sml 0 1 0 0 K*code
Printing Own machines list ...

      Keyboard_code

End of machines list
```

4.3.4 Reading Information on the Components

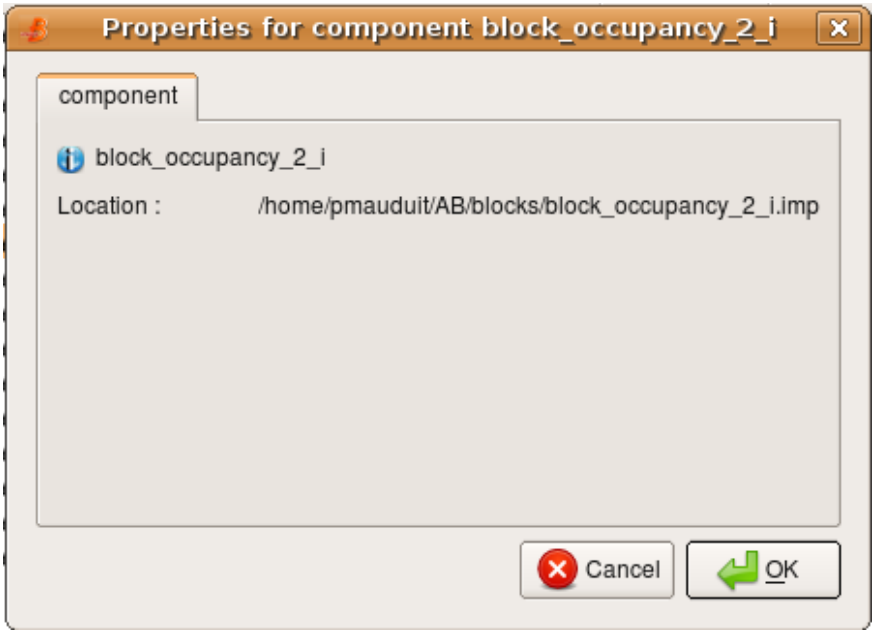
Description

This function displays the following properties of components:

- The entire access path of the B source file,
- The type of the component: machine, refinement or implementation.

Graphical user interface

In order to obtain information on a component, you have to select it and click on the *Component -> Properties* menu. A dialog like presented on the screenshot below appears then :



The distinction on the type of the component is done by the blue icon shown ib the interface.

Command-line interface

To get information on a component called `comp_name`, a project should be opened, then type the following command:

```
infos_component comp_name
```

or

```
ic comp_name
```

The properties are displayed as follows:

```
MACHINE         --> Main
LOCATION        --> /home/project/spec
OWNER           --> user1
```

4.3.5 Editing a Component

Description

This function allows you to edit the source file corresponding to a component.

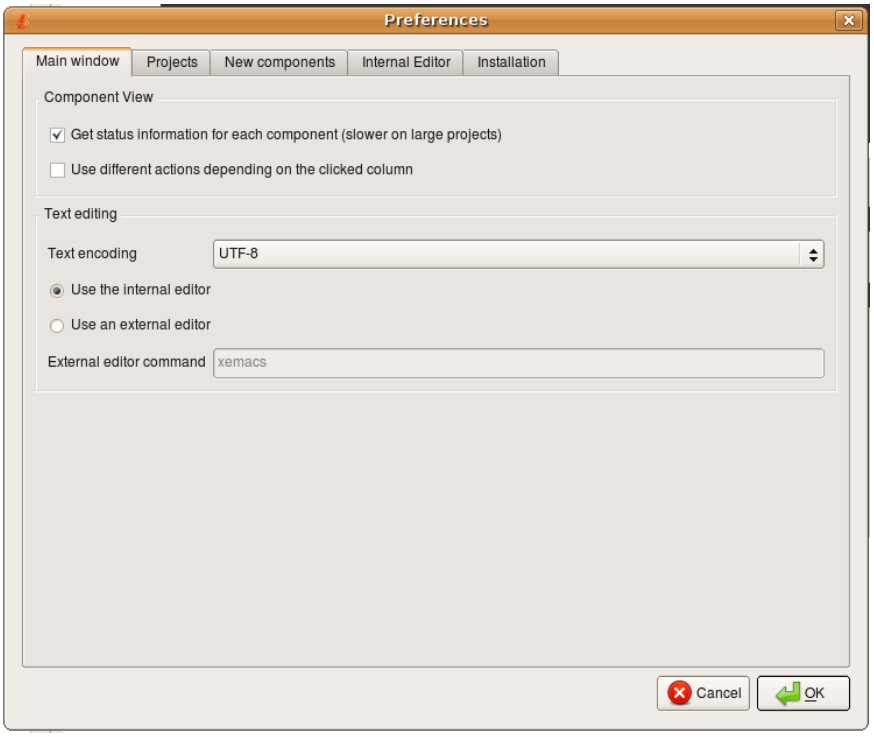
By default, the editor is the Atelier B internal editor, but this can be customized by the editor of your choice.

Graphical user interface

In order to edit a component in the GUI, by using the internal editor, you have to double-click on the

component. The editor is then launched. You can also use the *"Component -> Edit"* menu, or the keyboard shortcut *"Ctrl+E"*.

You can custom the editor configuration in the main Atelier B configuration. To do so, click on the *"Atelier B -> Preferences"*, and to configure your own editor. If you select an external editor, you will have to give a path to the path, as shown below :



Command-line interface

The user interface is started up, you already have opened a project.
To edit the `comp_name` component, a project should be opened, then type the following command:
`edit comp_name`
or
`e comp_name`

Usable parameters

*ATB*OPT_ TOOLS_ <SYSTEM>*Editor_ Path*
Positioned at the Atelier B set up
Access path to the text editor.

4.3.6 Restoring a Component

Description

This function is used to restore B source files or definitions files from an archive created using the function described in sub-section #4.2.4.

Restore is performed in the current project.

If the restored component is not in the current project, the function add it automatically. If the component is already present in the project, it is replaced by the restored component.

Graphical user interface

This command is not available in the GUI. However, you can restore the component's sources by hand, by following these instructions :

1. Uncompress the archive file,
2. Open in the GUI the given project,
3. Click on the *"Project -> Add components"* menu, and select the components source files that you want to add to your project.

Command-line interface

The user interface is started up, you already have opened a project.
To restore a component, use the `get_list_from_archive` command, followed by the `restore_source` command.

The `get_list_from_archive` command displays the list of files in a project archive:

```
bbatch 2> get_list_from_archive /home/project/tarPROJ.arc
x /tmp/atelierb.tar, 486 bytes, 1 tape blocks

Printing Components in archive file
        tarPROJ ...

      Acq_1.mch
      Arithmetic_1.mch
      Arithmetic_2.imp

End of List
```

The `restore_source` command performs the restore. For example, to restore the `Acq_1.mch` component in the `MyProj` project, you must type:

```
bbatch 3> restore_source /home/projet/tarPROJ.arc Acq_1.mch
x /tmp/atelierb.tar, 486 bytes, 1 tape blocks
x spec/Acq_1.mch, 342 bytes, 1 tape blocks
```

5 Applying the B-method

5.1 Presentation

To develop programs using the B method, Atelier B proposes a set of commands allowing:

- syntax and type checking of components,
- automatic generation of proof obligations (PO),
- automatic demonstration of POs,
- interactive demonstration of POs that are not automatically demonstrated,
- translatable language checking,
- translating implementation into one of the following programming language (C, C++, ADA or HIA).

The presentation of these commands assumes that the reader is familiar with the B method.

This manual therefore only covers the implementation conditions for the functions listed previously and not their aims in relation to the method.

In the basic version of Atelier B, translation to standard computer languages (ADA or HIA) is not included. The ADA or HIA translators must be installed separately.

5.2 Managing workspaces

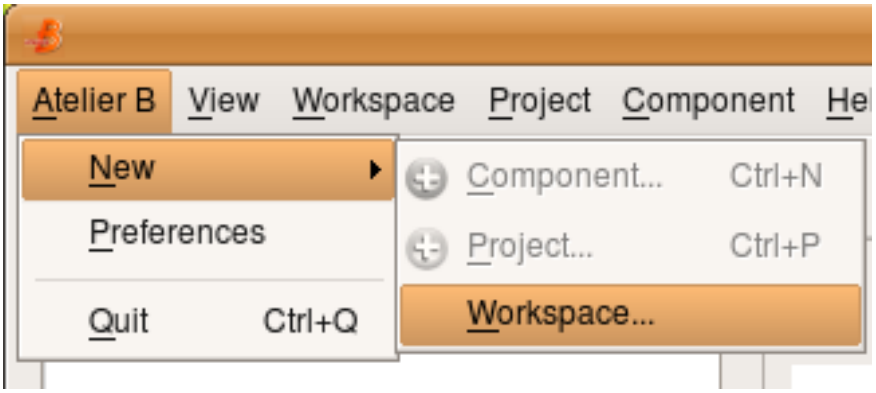
This section is useful only if you intend to use the GUI, since workspaces do not exist into the command-line interface.

While using the Command-line interface, you need to parameter a specific configuration. This is why the notion of workspace appear in the current version of the GUI : a workspace is a kind of configuration for the Command-line interface which will be used by the GUI.

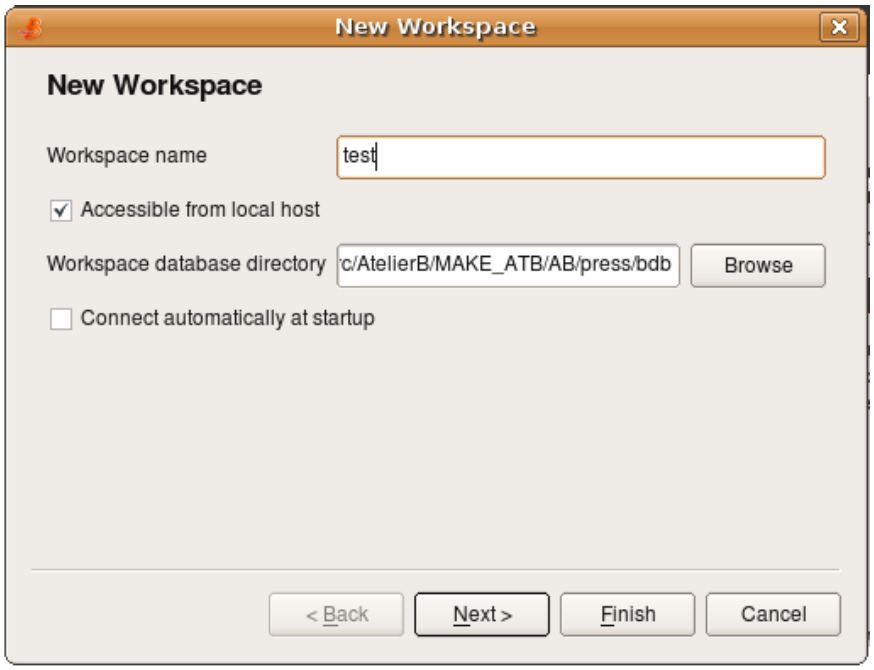
After having installed the Atelier B, a default workspace is automatically created and allows you to directly begin to develop. By the way, it is possible to tweak your configuration by creating a new custom workspace.

5.2.1 creating a Workspace

The workspace creation can be done into the GUI via the following menu:

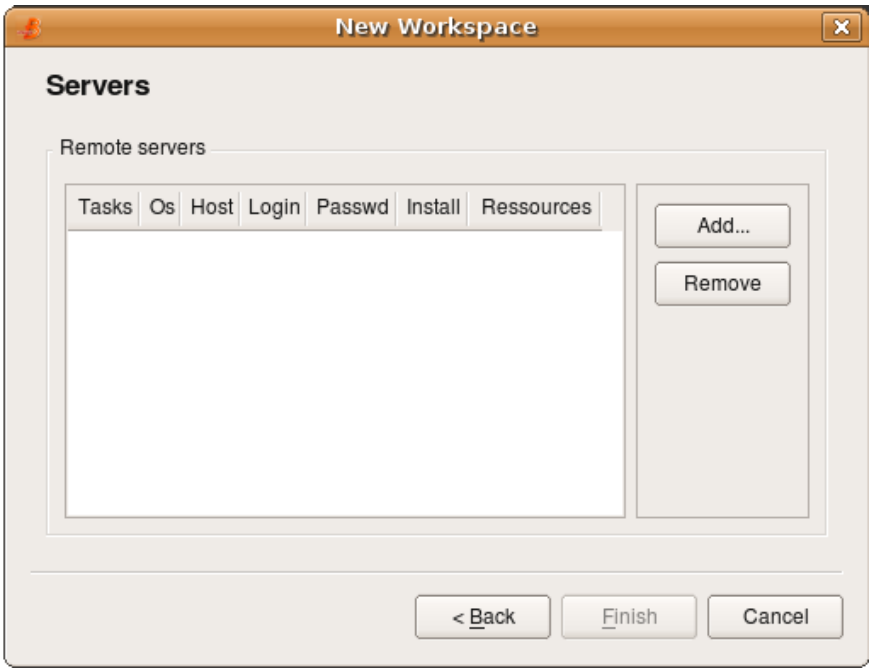


This menu will trigger the display of a workspace creation wizard, asking you in a first time to enter a name and a working directory:



Do not forget to check the options *"Accessible from local host"* if the workspace is intended to be local (i.e. if the path is available locally on the workstation), and if you want to open automatically this workspace at launch (*"Connect automatically at startup"* option).

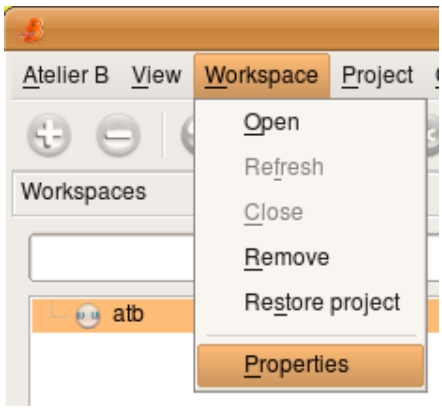
If you work on UNIX-compatible systems, the Atelier B integrates parallelization capabilities on remote computers, via the use of SSH. Then, you are able to specify remote machines on a second page of the wizard. This step is not necessary if you specified a local configuration, but is mandatory if the newly created workspace is intended to be a *"remote"* one.



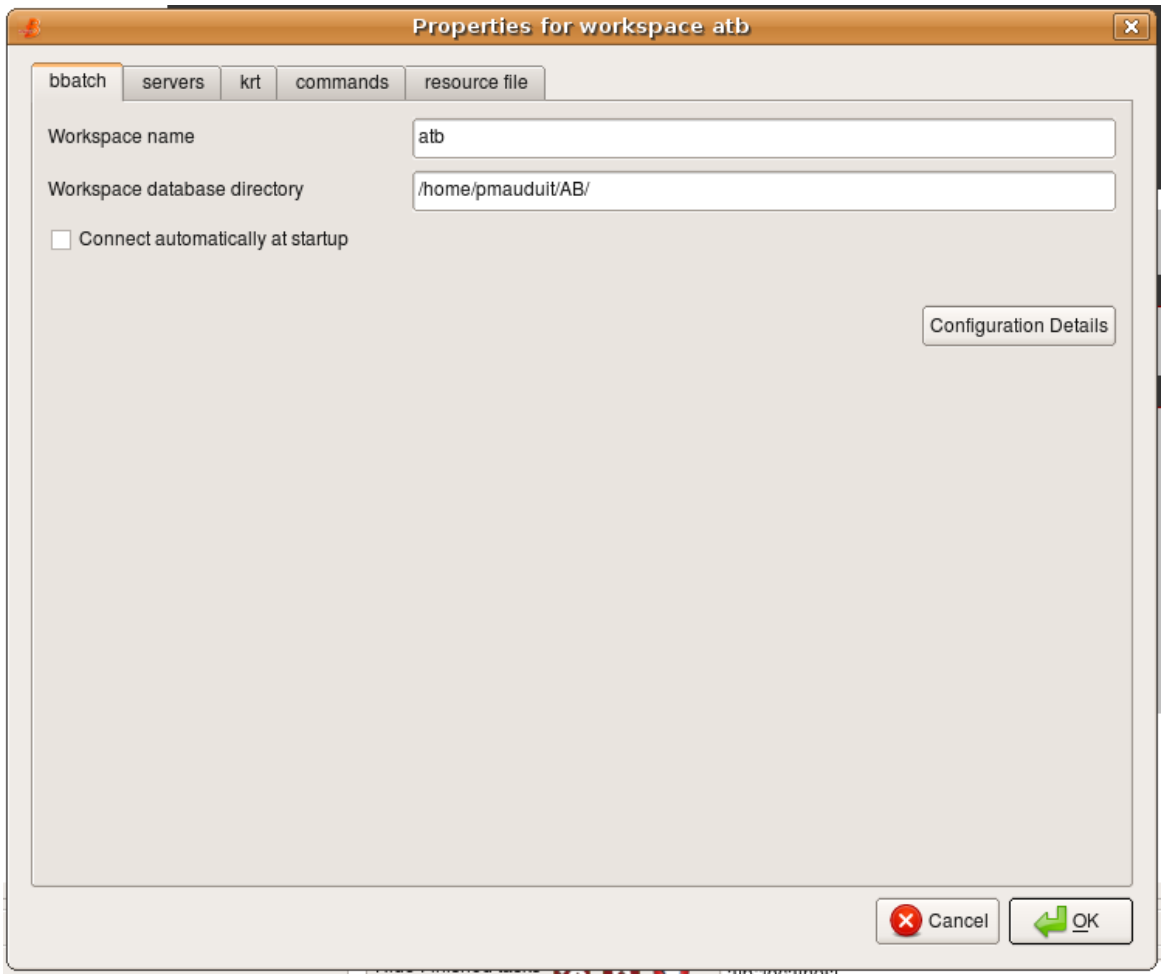
Once the wizard ended, you are now ready to manage B-projects.

5.2.2 Workspace Modifications

It could be interesting for the user to modify the workspaces properties. This is possible by triggering the *"Workspace -> Properties"* menu (or via a right-click on the desired workspace).



You will then get access to the following configuration dialog:



Here, we find back the properties already seen while creating a workspace, excepted the remote or local aspect. The *"Configuration Details"* button will give you information on the version of the different tools installed.

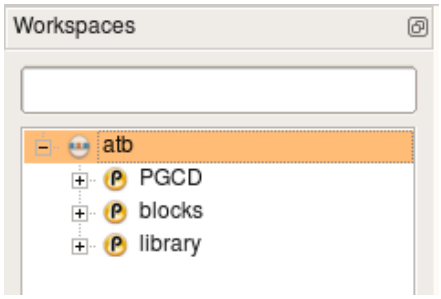
- The *"Servers"* tab allows you to specify remote servers, which can execute in parallel different tasks on B-Projects,
- The *"krt"* contains some controls about the Atelier-B kernel (the krt binary),
- The *"Commands"* tab allows you to configure some tools : the Typechecker, xref, the PO-Generator, the prover, the predicate-prover, the lemmas prover, and the "Delta Component",
- The *"Resource file"* tab lets you the ability to tweak your resource file, and is meant to be used by advanced users. Please consult the paragraph 8 for more information.

5.2.3 Deleting a Workspace

The deletion of a workspace is simply done by triggering the *"Workspace -> Remove"* menu. This will not suppress any file related to your projects from your hard drive.

5.2.4 Opening and Closing a Workspace

A double-click on a workspace, or a click on the *"Workspace -> open"* menu will open the desired workspace ; the icon is then changing, and the available project is displayed:



In order to close a workspace, use the *"Workspace -> close"* menu. The link between the GUI and the Command-line interface is then closed, and the Workspace icon is being changed:

5.3 Syntax analysis and Typecheck

Description

This function combines the syntax analysis and the type checking of B components.

The syntax checking ensures that the sources for the selected machines comply with the B language syntax. On this subject, the reader can refer to the *B language - Reference Manuel*.

Type checking controls:

- identifier conflicts,
- typing rules,
- missing declarations,
- language restrictions,
- visibility rules,
- etc ...

This type check is necessary for the PO generation.
The type check of a component is automatically applied to all the components "required" by the current component, through the following links SEES, USES, INCLDES, IMPORT, EXTENDS, REFINES. This type check on the required components is only applied when necessary, i.e. if a component change was made since the last type check.

Modifications of "form" (comments, spaces, ...) are not taken in account.

Syntax errors are displayed in an error window and in the start-up window.

They are displayed as following:

```
<file>:<line number>:<column number> <error description>
Example:
AA.mch:6:17 Sequential (';') substitution is not allowed in a specification
```

Line and column numbers show exactly the location in the source file where the error occurs.

Semantic errors are displayed in the Atelier B start-up window.

They are displayed as following:

```
Type checking machine AA

Loading referenced_machines
Checking name_conflicts
```

```
Checking constraints clause
...
    Checking INVARIANT clause
Error: 1+2 in ( aa: 1+2 ) should be a set
Error: Variable aa has not been typed
  Checking operation b_ask_code
Checking operation b_code_typed
    No information saved for AA

End of Type checking
```

The type checker displays an information message for each processed clause. The *Type checker - Error messages manual* describes in detail all the error messages generated during this phase.

Graphical user interface

The GUI is launched and a project is opened.
To do a syntax analysis and a typecheck on components, you have to:

1. Select the desired components into the components list,
2. Click on the *"Typecheck"* button. each component is processed successively.

   If there is a syntax error, an error window will be displayed. This window contains the description of the detected errors.

   If there is a semantic error on one of the components, a warning window will be displayed. In this case, check the type of error according to the messages displayed in the start-up window. If a syntax error occurs, the error is reported with a description into the errors view.

Remark : You can interrupt the treatment by using the function described in chapter 5.15 on page 81.

Command-line interface

The user interface is started up, you already have opened a project.
To perform a syntax analysis and a type check on component `comp_name`, type the following command:
`typecheck comp_name`
or
`t comp_name`

The information and error messages from the type checker are displayed in the start-up window.

Usable parameters

*ATB*BCOMP*Allow_ANY*
FALSE
Allow or not the ANY substitution in implementation.

*ATB*BCOMP*Allow_Becomes_Member_Of*
FALSE
Allow or not the "becomes member of" substitution in implementation.

*ATB*BCOMP*Allow_Becomes_Such_That*
FALSE
Allow or not the "becomes as" substitution in implementation.

*ATB*BCOMP*Allow_CHOICE*
FALSE
Allow or not the CHOICE substitution in implementation.

*ATB*BCOMP*Allow_LET*
FALSE
Allow or not the LET substitution in implementation.

*ATB*BCOMP*Allow_Parallel*
FALSE
Allow or not the Parallel substitution in implementation.

*ATB*BCOMP*Allow_Pre*
FALSE
Allow or not the PRE substitution in implementation.

*ATB*BCOMP*Allow_Read_In_Values*
FALSE
Allow or not using previously valued constants in the VALUES clause.

*ATB*BCOMP*Allow_SELECT*
FALSE
Allow or not the SELECT substitution in implementation.

*ATB*BCOMP*Tab_Width*
8
Number of character needed to obtain a tabulation.

5.4 Automatic refinement

Description

A functionality of the Atelier B permits to generate refinements automatically, based on the use of the *Bart*

Graphical user interface

The GUI is already launched, and a project is opened.

In order to use this functionality, click on the desired component, then trigger the menu *"Component -> Automatic Refinement"*.

The newly created components are automatically attached to the project, and a comment is added to their sources:

```
/* <nom_comp>
 * This file has been generated by BART (B Automatic Refinement Tool)
 * Generated the ION10/2008
 * DO NOT EDIT
 */
```

Command-line interface

The interface is launched, a project is opened.

To use the automatic refinement function on the component *nom_ comp*, you have to type in the following command:

```
bart <nom_comp>
```

5.5 Generating Proof Obligations

Description

This function produces the proof obligations of a component. The component must be type checked (refer to the section #5.3).

The proof obligations are defined by the B method. They depend on the level of the software development:

- In machines, the selected mathematical model must be consistent.
- In the following steps, you must prove that refinements keep the properties of the previous step model.

The document *Proof Obligations _ Reference Manual* describes the PO in a theoric way.

In theory there is a proof obligation for initialization and a proof obligation for each of the operations.

In practice, these obligations can be "large" and complex formulas. The *Proof Obligations Generator* function split the theoretical PO into many simpler PO. Some PO are so easy to prove that the PO Generator can prove them by itself. In return, the initially foreseen number of formula increases. Some PO defined as obvious are automatically eliminated by the tool.

Before generating the proof obligations for a component, Atelier B ensures that the component is type checked. Otherwise, the type check is automatically performed.

Generating proof obligations creates four files in the PDB:

- the `comp_name.po` file contains the PO of the `comp_name` component.
- the `comp_name.opo` file contains the obvious PO of the `comp_name` component.
- the `comp_name.pmi` file contains the status of the proof obligations (proved/not proved) as well as the interactive demonstrations.
- the `comp_name.stc` file contains a description of the component.

If the Differential option is used, and if the PO of the component have already been generated at least once, Atelier B compares the component with the description saved in the `comp_name.stc` file.

For each operation, and for the initialization, it generates the PO only if one of the information occuring in their construction has been modified. Otherwise, it copies the PO from the ancient files.

If the full option is used, the Atelier B generates all the PO again, even if they haven't been modified.

After the new PO generation, and if they have already been modified before, they are automatically compared to the ancient ones, in order to keep the associated demonstrations. The comparison rule is the following one:

A P.O.B deducts itself from a P.O.A if they have the same goal, and if the assumption of B contains the assumption of A.

If a PO can deduct itself from one or several ancient PO, it receives by preference order:

- the demonstration of the ancient PO with the same number if this PO would end,
- the first one of the ancient PO demonstrations proved of the same clause,
- the first one of the ancient PO demonstrations proved of a different clause,
- the demonstrations of the ancient PO with the same number wich was not ending.

If a new PO can't deduct itself from an ancient one, but if its number and pertenancy clause do exist, it receives the demonstration of the ancient PO with the same number and clause. In this way, the user retrieves his demonstation even if he renamed some identificators.

Thanks to this mecanism, the user can keep the interactives demonstrations (and automatics) if the PO are the same.

The messages of PO generator are displayed in the start up window as following:

```
Generating proof obligations of Machine B_Keyboard_code

    Initialisation :
......
proof obligations:          3
obvious proof obligations:  3

    b_input_code :
..
proof obligations:          0
obvious proof obligations:  2

    3 proof obligations generated

    11 obvious proof obligations generated

Generation complete

Normalising...
    b_check_code: unchanged

Merging...
Done
```

For each clause present in the component, Atelier B displays the number of proof obligations to be proved (`proof obligations:`) and the number of obvious proof obligations deleted (`obvious proof obligations:`).

Atelier B displays a character "`.`" each time a new PO is generated.

The PO clauses that have been copied are listed at the end of generation.

Graphical user interface

The GUI is already launched, a project is opened.

To generate the Proof Obligations on components, you have to perform these following steps:

1. Select the components into the components list,
2. Click on the *"PO Generate ..."*. If the POs have been generated for a given component, the Atelier B will ask the user if it is necessary to execute a command in *full mode*.

   In case of errors on a component, a message is added into the errors view.

Command-line interface

The user interface is started up, you already have opened a project.
To generate the proof obligations for component `comp_name`, with the *Differential* option, type the following command:
`pogenerate comp_name 1`
or
`po comp_name 1`

To generate the PO for component `comp_name`, with the *full* option, type the following command:

`pogenerate comp_name 0`
or
`po comp_name 0`

The option by default proposed by Atelier B is the *Differential* mode.

Usable parameters

*ATB*POG*Generate_ Obvious_ PO*
Positioned at Atelier B installation.
Determine wether the Atelier B will generate or not the obvious PO.

5.6 Displaying Proof Obligations

Description

Atelier B provides two methods for displaying proof obligations:

- Using the *PO Viewer*
- Using the interactive prover

Using the interactive prover is recommended in the following cases:

- complex assumptions: in this case the search functions of the interactive prover are required to analyse these proof obligations.
- the number of proof obligations is high. The PO would rather be displayed one by one which is impossible using the *POViewer*.

The *Interactive Prover - User's Manual* sub-section 5.4 describes the different methods available for viewing proof obligations.

This sub-section describes the use of the *POViewer*.

This tool enables:

- displaying the proof obligations of a component clause by clause,
- displaying obvious proof obligations, i.e. those that were eliminated by the proof obligations generator,
- displaying and printing proof obligations using mathematical fonts via a word processing program (LaTeXor Word).

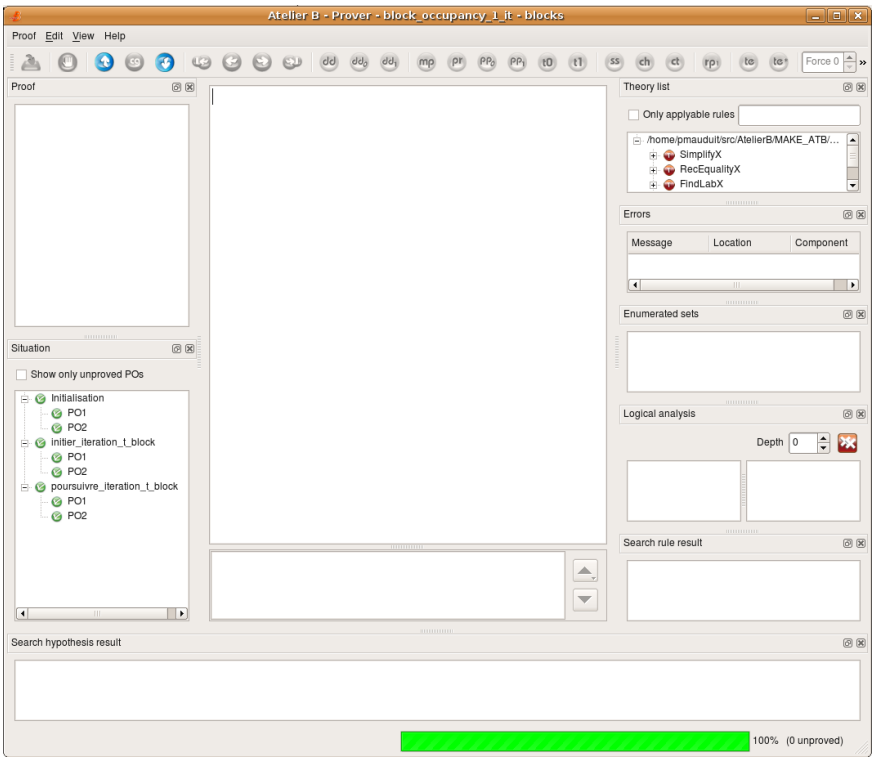
The proof obligations include comments that specify:

- the origin of the assumptions (for example: `Component invariant`),
- the theoretical justification of the proof obligation.
  In this case the comment refers to a sub-section in thex *Proof obligations - Reference manual.*

Graphical user interface

The GUI is supposed to be launched, a project is opened. To open the GUI of the interactive prover, it is necessary to click on the given component, and trigger the *"Component -> Proof -> Interactive Proof ..."* menu.

The following window will be then dispayed:



The Proof Obligations are visible into the *"Situation"* view.

Usable Parameters

*ATB*OPT_ TOOLS_<SYSTEM>*Latex_ Binary_ Directory*
Positioned at Atelier B installation.
Directory where it is possible to find the Latex binaries.

*ATB*OPT_ TOOLS_<SYSTEM>*Latex_ Viewer*
Positioned at Atelier B installation.
Name of the Latex viewer.

5.7 Automatic Demonstration

Description

This function automates, in the limit of its ability, the demonstration of the proof obligations for each B component.

The proof activity is essential to the B method. This is the reason why two manuals are dedicated to this subject:

- *Interactive Prover - User's Manual*
- *Interactive Prover - Reference Manual*

The Atelier B automatic prover has different levels of force. These forces are described in sub-section 3.1.1 of the *Interactive Prover - User's Manual*

The messages of the automatic prover are displayed in the start-up window as shown below:

```
Proving B_Delays

  Proof pass 0, still 3 unproved PO

    clause b_init_delas

        -+

End of Proof
  Initialisation  Proved 0  Unproved 0
  b_init_delay  Proved 1  Unproved 0
  b_stop_delay  Proved 0  Unproved 0
  b_delay_is_up  Proved 3  Unproved 0
TOTAL for B_Delays  Proved 4  Unproved 0
```

For each clause of the component, Atelier B displays a + each time a proof obligation is proved and a - each time the prover fails.

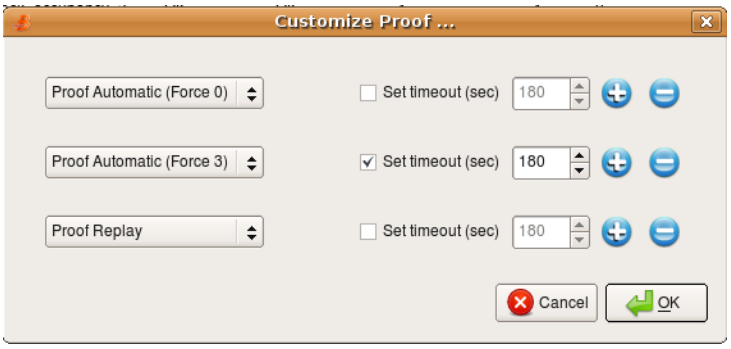
Graphical user interface

The GUI is already launched and a project is opened.

To launch the automatic prover on some components, you have to perform the following instructions:

1. Select the componets into the list,
2. Click on the *"Component -> Proof"* menu, and on the desired force.

You can in addition launch a customized proof, by triggering the *"Component -> Proof -> Customize Proof"* menu. The following window will then show up:



you can then add a programmed sequence of different forces to your components selection, and fully customize your proof.

You could interrupt the process, by defining a timeout.

Command-line interface

The user interface is started up, you already have opened a project.

To run the automatic prover on component `comp_name`, type the following command:

```
prove comp_name <force>
```
or
```
pr comp_name <force>
```

The `<force>` value could be:

0,1,2,3 for the different prover force levels,

-1 for "Fast" prover level.

-2 for the "Replay" option.

-3 for the "User Pass" option.

5.8 Interactive Demonstration

Description

The primary goal of this function is to allow the user to prove manually the proof obligations that were not proved automatically.

Graphical user interface

The GUI is launched, and a project is opened.

In order to launch the interactive prover, you have to follow these steps:

1. Click on the chosen component from the components list.
2. Click on the *"Component -> Proof -> Interactive Proof"* menu, or use the keyboard shortcut *"Ctrl+i"*.

From the prover GUI (see paragraph 5.6), you can perform a manual proof. The progress is indicated via a progressbar situated in the bottom right of the interface. In addition, the proof status is indicated into the *"Situation"* view. See the *interactive proof manual* for more information (accessible via the *"Help -> Help contents"* menu).

Command-line interface

The user interface is started up, you already have opened a project.
To run the interactive prover on component `comp_name`, type the following command:

```
browse comp_name
```
or
```
b comp_name
```

After typing this command the interactive prover prompt is displayed: `PRI >`.

You can then type the various interactive prover commands.

Type `qu` to quit.

5.9 Cancelling Demonstrations

Description

This function is used to cancel the demonstrations performed on a component in order to repeat interactive proofs.

The interactive demonstrations are not lost. Only the status of the proof obligations is changed.

Graphical user interface

The GUI is launched, a project is opened.
To cancel the component demonstrations, perform the following operations:

1. Select the components from the list of components.
2. Use the *"Components -> Unprove"* menu.

The return messages into the *tasks* view must then indicate the following label:

```
Unproving successful
```

Command-line interface

The user interface is started up, you already have opened a project.
To cancel demonstrations of the component `comp_name`, type the following command:

```
unprove comp_name
```
or
```
u comp_name
```

The following message is displayed in the start-up window:
`Unproving successful`

5.10 Checking the Translatable Language (B0)

Description

Before using the translators for exporting to standard computer programming languages, check that the language used in the implementations can be translated.

The constructions authorized in the implementations are described in the *B Language - Reference Manual*.

The error messages from the B0 checker are displayed in an error window and in the start-up window as following:

```
<file>:<line number>:<column number>(B0 check)<error description>
```

```
Exemple:
B0 Checking Machine B_Keyboard_code_1

B_Keyboard_code_1.imp:5:11 (B0 Check) binary expression is not a simple term
B_Keyboard_code_1.imp:5:19 (B0 Check) binary expression is not a simple term
B0 Check error in B_Keyboard_code_1
```

Column and line numbers allow an exact location of the place where the error was detected.

When the component is correct, the following message is displayed in the start up window:

```
B0 Checking B_Keyboard_code_1
B0 Checking B_Keyboard_code_1 successful
```

`remark`: To use the HIA translator, it is necessary to type the array with concrete constants. By default, the B0 checker indicates that the concrete constant is not implementable. In that case, to pass over the B0 checker, the following resource has to be positioned:

ATB*BCOMP*Enable_Typing_Identifiers: TRUE

It is easier to save this resource in the project resource file of the projects wich will be translated into the HIA language.

Graphical user interface

The user interface has already been started up, a project is opened.

To check the translatable language on the components, perform the following operations:

1. Select the components from the list of components.
2. Use the *"Component -> B0 check"*, or directly the keyboard shortcut *"Ctrl+B"*.
   The result of all checks is displayed, component by component, in the start-up window.

   If there is an error on one of the components, an error message is displayed in the *"error"* view. A double-click on the error will then open the internal editor (if configured as the default editor) at the error location.

Command-line interface

The user interface is started up, you already have opened a project.
To B0 check component `comp_name`, type the following command:
`b0check comp_name`
or
`b0c comp_name`

Usable Parameters

*ATB*COMP*Disable_ Array_ Compatibility_ Check*
FALSE
Perform or not compatibility checks of array indexes.

*ATB*COMP*Disable_ Concrete_ Constants_ Type_ Check*
FALSE
Perform or not type checks of concrete constants.

*ATB*COMP*Disable_ Expression_ Syntax_ Check*
FALSE
Perform or not expression syntax checks .

*ATB*COMP*Disable_ Formal_ Params_ Type_ Check*
FALSE.
Perform or not type checks of formal parameters .

*ATB*COMP*Disable_ Variables_ Initialisation_ Checker*
FALSE
Perform or not variables initialisation checks.

*ATB*COMP*Disable_ Locale_ Variables_ Type_ Check*
FALSE
Perform or not variables type checks.

*ATB*COMP*Disable_ Operation_ Input_ Parameters_ Type_ Check*
FALSE
Perform or not operation input parameters type checks.

*ATB*COMP*Disable_ Operation_ Output_ Parameters_ Type_ Check*
FALSE
Perform or not operation output parameters type checks.

*ATB*COMP*Disable_ Parameters_ Instanciation_ Check*
FALSE
Perform or not machine parameters instanciation checks.

*ATB*COMP*Disable_ Predicate_ Syntax_ Check*
FALSE
Perform or not predicate syntax checks.

*ATB*COMP*Disable_ Valuation_ Check*
FALSE
Perform or not VALUES clause checks.

*ATB*COMP*Enable_ Typing_ Identifiers*
FALSE.
Variables of array, record or interval type must be typed with an identifier if this resource is TRUE (special case for HIA translations) .

5.11 Project Checking

Description

This function, only available via the command-line interface, performs checks on all the project components.

The rules checked by this function are:

- a machine can only be imported once in a project,
- a seen machine must be imported by a project component,
- the SEES clause must be transversal to a component,
- the dependency graph must not contain cycles,
- the names of project components must be differents (an upper/lower case difference is not enough).

These checks are described in the *B language - Reference Manual*.

These checks are required to translate the project. They are run automatically by Atelier B before project translation.

Some of these checks are performed automatically before the syntax analysis and the type check of components in order to warn the user as soon as possible.

The user can also perform these checks on demand by following the procedures described below.

Command-line user interface

The user interface is started up, you already have opened a project.
To perform the checks on this project, type the following command:
`project_check comp_name`
or
`pchk comp_name`

The parameter of this command is the name of the implementation that is the project entry point.

5.12 Translating

Description

In the basic version of Atelier B, only the C translator is shipped (named *ComenC*).

This function translates project implementations into different languages.

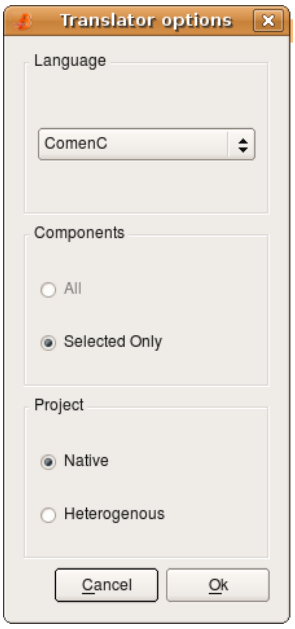
For further information, refer to the translators user's manual.

Graphical user interface

In order to translate a component using the `ComenC` translator, perform the following instructions:

1. Select the desired component into the list of components,
2. Trigger the *"Component -> Generate Code"* menu.

The following dialog will then appear:



After having selected the desired options, click on the *"Ok"* button to confirm and launch the translation process.

Command-line interface

To translate an implementation `imp_name` into C language, you have to type in the following command:

```
ComenCtrans imp_name
or
b2c imp_name
```

5.13 Applying a Tool to all the Components of a Project

Description

The *Make project* function, only available with the command-line interface is used to perform, on all the components of a project, the following operations:

- syntax analysis and types check,
- generating proof obligations,
- proof,
- checking the translatable language (B0),

This function takes into account the links between project components; its operation is similar to the UNIX "make" function.

This function proposes two options:

forced mode The requested operations will be performed regardless of the state of the components. For example, if a component is already in *TypeChecked* state and the user requests *Forced Make* on the project, then the type check will be repeated on this component.

normal mode The requested operations are only performed if necessary.

Warning: When a *Forced Make* is requested on the project, only the requested operation will be systematically repeated on all the project components.

For example: If you request a *Forced Make* for the *POgenerate* operation, then the generation of proof obligations will be repeated on all the project components. The type check will not be repeated.

Command-line interface

The user interface is started up, you already have opened a project.
To perform an operation on all the project components, type the following command:

```
make_all operation force
```
or
```
m operation force
```

The `operation` parameter can take one of the following values: `typecheck`, `pogenerate`, `b0check`, `prove`.

The `force` parameter must equal `0` in normal mode and `1` in forced mode.

If the requested operation is `prove`, you must give a third parameter which is the prover force level to apply (-3,-2,-1,0,1,2 or 3. Refer to sub-section #5.7).

5.14 Updating a Project

Description

When several users are simultaneously working on a large scale project, modifying a component can have effects on several other project components (on all the components linked to this component).

The *Remake project* function is used to update all the components of a project.

It takes into account the dependencies between the project components and the status of each component.

This function "redoes" for each component, all the actions that have already been performed at least once.

This function offers two options:

forced mode Operations that have already been performed at least once will be repeated, regardless of the status of the components. For example, if a component is already in the *Type Checked* state and the user requests a *Forced remake* on the project, then the type check will be redone on this component.

normal mode The operations that have been performed at least once in the past will be repeated only if necessary.

Command-line interface

The user interface is started up, you already have opened a project.
To update all the project components, type the following command:

```
remake force
```
or
```
r force
```

The `force` parameter must equal `0` in normal mode and `1` in forced mode.

5.15 Tools interruption

Description

This function permits the interruption of some actions.

According to the executed action, this interruption is either automatic or manual.

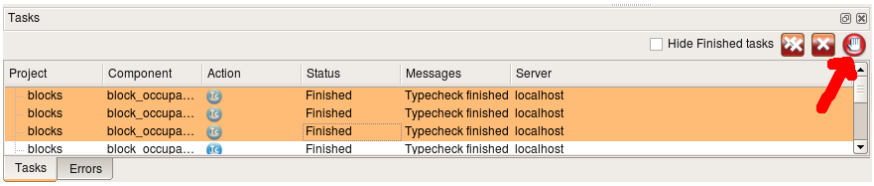
- The manual interrupt works with the following actions :
  - Syntax analysis and type checking,
  - Generation of proof obligations,
  - Automatic demonstration,

  It offers different possibilities according to the action that is executing.
- The automatic interrupt is used with the automatic demonstration.
  A timeout can be defined for the automatic demonstartion in order to try some prove tactics which can loop or take too much time.

Graphical user interface

The GUI is already launched, a project is opened.
After having selected one or more components, click on the button corresponding to the action of your choice : *Typecheck, PO Generate, Proof, ...*

When the action begins, it appears into the *tasks* view. It then possible to interrupt it by clicking on the following icon:



5.16 Dependencies Management

Description

For each of the actions described in this chapter, Atelier B checks that this action has not already been performed on this component.

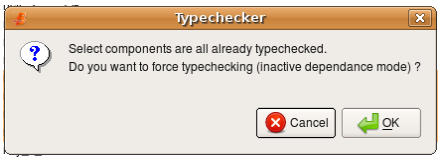
The action is only carried out if the component, or the components it depends on, have been modified.

If this is not the case, the following messages are displayed :
Component <comp_name> is already Type Checked
or
Proof obligations already generated for <comp_name>

The functions described below permit you to "force" the accomplishment of the action.

Graphical user interface

When some actions seem to be performed uselessly, the interface detects it and asks then the user for a confirmation. In example, the following question could be asked to you after a *"Typecheck"* :



Command-line interface

The user interface has already been started up.
To stop the management of dependencies and therefore be able to force an action, you must type the following command:

`disable_dependence_mode`
or
`ddm`

You can then type your command.

For example : `typecheck AA`.

To bring the management of dependencies back into function, you must type the following command:

`enable_dependence_mode`
or
`edm`

6 Analysing B developments

6.1 Presentation

"*Analysing a B development*" is a set of commands used to obtain information on the components of a project.

The analysis commands are used to:

- determine the status of a project (syntax checked, proven, etc.),
- determine the proof status of a component (number of proof obligations per operation, etc.),
- create a dependency graph between the components.

6.2 Project status

Description

This function is used to create a summary table that provides information on all the components of a project.

It is used to determine project progress.

This function takes into account the dependencies between project components. If some components are not yet present in the project, a warning message is displayed. The results table is displayed in the Atelier B start-up window. This table can also be used by the Atelier B documentation tools (refer to sub-section #7).

The columns in the generated table indicate:

TC (or Typechecked) An *OK* value means that the syntax analysis and type check of the component and all the components it depends on were successfully performed.

POG (or PO Generated) An *OK* value means that the proof obligations were generated for the component.

nPO (or Proof obligations) This column contains the number of non obvious proof obligations of the component.

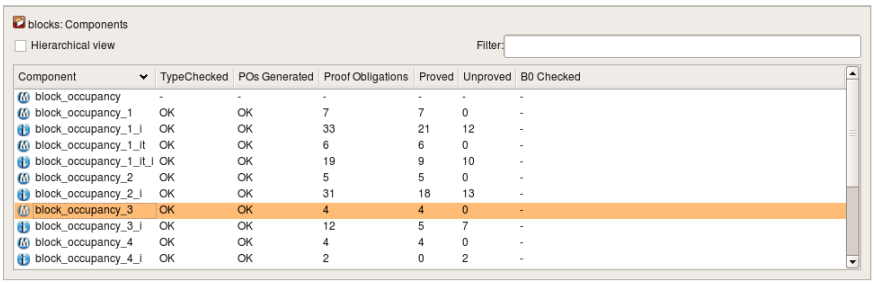
nUn (or Unproved) This column contains the number of proof obligations for the component that have not yet been proved.

%Pr This column contains the percentage of proof obligations already proven. This percentage does not take into account the obvious POs.

B0c (or B0 checked) An *OK* value means that the B0 check was performed successfully on this component.

Graphical user interface

Once a project is opened, the status is directly available on the components view. An example is given on the following screenshot:



We can notice that the columns may vary depending on the use of the interfaces and the user preferences.

Command-line interface

The user interface is started up, you already have opened a project.

To obtain the project status, type the following command:
`status_global`
or
`sg`

6.3 Component Status

Description

A component with the B method applied on pass through several states:

Modified after modification of the component source,

Parsed after a syntax analysis of the component,

TypeChecked after type checking of the component,

POGenerated after generating of component proof obligations,

AutoProved after an automatic or interactive demonstration of all the proof obligations of the component.

These states are exclusive. A component loses its *Modified* state as soon as it is successfully type checked.

This function displays the component state as shown below:

```
Printing the status of QUERY

    QUERY TypeChecked /home/project/spec/QUERY.mch

End of Printing the status
```

If the component is the *POGenerated* status, the function displays a table showing more precise information on the component proof obligations.

Example:

```
Printing the status of FILE_BUFFER_1

    FILE_BUFFER_1 POGenerated /home/projet/spec/FILE_BUFFER_1.mch
+-----------------------+-------+------+-------+-------+-----+
|                       | NbObv | NbPO | NbPRi | NbPRa | %Pr |
+-----------------------+-------+------+-------+-------+-----+
|   Initialisation      |    3  |   0  |       |       |     |
|   load_buffer         |   12  |   3  |   0   |   3   | 100 |
|   create_record       |    4  |   1  |   0   |   0   |   0 |
|   not_in_buffer       |    7  |   0  |       |       |     |
|   mod_buffer          |    8  |   0  |       |       |     |
|   val_buffer          |    8  |   0  |       |       |     |
|   size_file           |    5  |   0  |       |       |     |
+-----------------------+-------+------+-------+-------+-----+
|   FILE_BUFFER_1       |   47  |   4  |   0   |   3   |  75 |
+-----------------------+-------+------+-------+-------+-----+

End of Printing the status
```

The columns in this table show, for each operation on the component:

NbObv This column contains the number obvious proof obligations of the operation. These proof obligations are eliminated automatically by the generator of proof obligations.

NbPO This column contains the number of not obvious proof obligations of the operation.

NbPRi This column contains the number of proof obligations proved by the interactive prover.

NbPRa This column contains the number of proof obligations proved by the automatic prover.

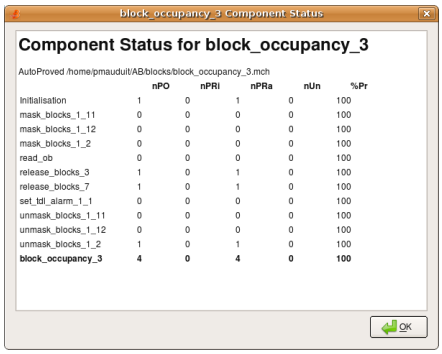
%Pr This column contains the percentage of proof obligations on the operation already proved. This percentage does not take in account POs eliminated by the proof obligation generator.

The last line in the table, (`TOTAL`) sums up each information for all operation of the component.

This information can be included in the documents automatically generated by Atelier B (refer to sub-section #7).

Graphical user interface

The GUI is aleardy launched, a project has been opened. To obtain information on a component, you have to click on the given component, and trigger the *"Component -> Status"* menu, or either use the *"Ctrl+s"* keyboard shortcut. A dialog is displayed, as on the following screenshot:



Command-line interface

The user interface is started up, you already have opened a project.
To obtain the status of the `comp_name` component, type the following command:
`status comp_name`
or
`s comp_name`

6.4 Dependency Graphs

Description

This command is only available if the workstation has the *GraphViz* free package tool installed (visit `http://www.graphviz.org/` for more information).

A dependencies graph is provided for the selected component or for the whole project.

The search for dependencies is recursive. Therefore if component X depends on component Y (for example via an "IMPORTS" link), and component Y depends on component Z (for example via a "SEES" link), then components X, Y and Z will be present in the graph.

The components are grouped by module. A module contains a machine, its refinements and its implementation.

Given the complexity of the projects, a number of options are available for filtering the links and the modules displayed. The user can choose between several options in order to filter the links and the modules being printed ; Here are the different options:

Components filtering: The user can choose between several options for the components. The available options are:

All: All the project components are present.

Selected only: Only the selected component is displayed.

Selected and transitively linked: Only the selected component and all its refinements and abstractions are displayed.

Link filtering: The user can choose the types of links (SEES, IMPORTS, ...) displayed.

Graph direction: The user may choose to do a graph:

ascending: In this case, the components that are dependent on the selected component are displayed. This shows the impact of a modification to this component on the other project components. All the component incoming links are displayed.

descending: In this case the component linked to the selected component are displayed. All the components outgoing links are displayed.

Library components filtering: The user may choose between several options for the components linked to the project and that are in a library. The options available are:

Show All: In this case the components present in libraries are handled in the same way as the other components. If these components depend on other components, they will also be included in the graph.

Show: In this case only components directly linked to project components will be displayed. If these components depend on other components, they will not be displayed.

Group: In this case all the components in the same library are combined in the same node of the graph.

Hide: In this case the components present in libraries will not be displayed.

For all components present in libraries, the library name is given in brackets.

Components not linked filtering: The user can choose to not see the isolated components, in other words, the components that are not linked to other components.

Instanciation graph: The user can choose to visualise only the instanciation graph, in other words, only the IMPORTS links between the different modules.

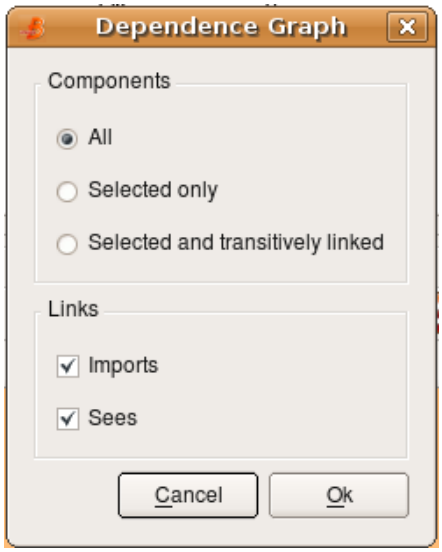
Dependency graphs can be included in the documentation automatically generated by Atelier B (refer to section #7).

Graphical user interface

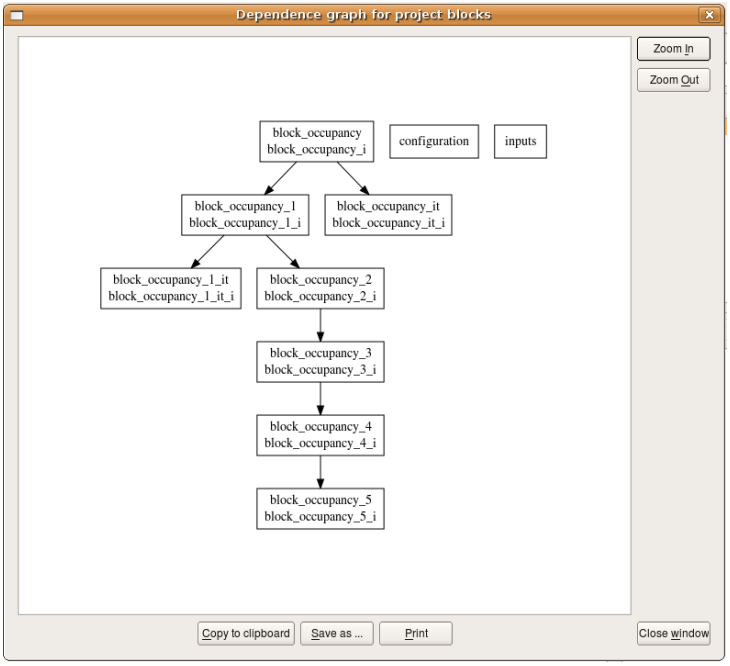
The GUI is already launched, a project is opened. To obtain a dependency graph, the user have to perform the following operations:

1. If you want a dependency graph about a specific component, select it from the components list,
2. Trigger the *"Component -> Dependence Graph"* menu,
3. Select the desired options on the dialog, then click on the *"Ok"* button.

Below is a screenshot of the dialog which allows you to select the different options:



The window presenting the graph will finally appear:



Command-line interface

The user interface is started up, you already have opened a project.
To obtain a dependency graph, use the following command: `project_status`
or
`ps`.

This command uses seven parameters:

1. The name of the component or the "`*`" value to obtain a graph for the entire project.
2. The option on components: `A` for *All*, `S` for *Selected only*, `G` for *Selected and transitively linked*.
3. The library option: `A` for *Show All*, `S` for *Show*, `G` for *Group* `N` for *Hide*.
4. The direction of the graph: `U` for *up*, `D` for *down*.
5. The types of links to display. The links are in the following order: EXTENDS, IMPORTS, INCLUDES, SEES, USES. The value `0` deletes the displaying of the link. Example: To display only IMPORTS and SEES: `01010`.
6. The option on the components not linked: `1` to hide the isolated components, `0` to show them.
7. `1` to see only the instanciation graph, otherwise, `0`.

Example: To display the project graph with all the links, type the command:
`project_status * A D 11111 0 0`

6.5 Operation call graph

Description

This command is available only if the workstation has the *DaVinci* tool installed, and only by using the Command-line interface.

The operation call graph permits to visualise the cascades of operation calls in the OPERATIONS, LOCAL_OPERATIONS and INITIALISATION clauses of a B component. This type of graph is useful during the proof phase, as it allows to better understand where the elements of an operation PO come from.

To distinguish the specifications and implementations of local operations, the name of the implementation of a local operation is preceded by `refinement_of_` in this graph.

The operation call graph is provided for some selected components, for these components and all those they depend on or for all the components of a project.

The operation call graph can be included in the documentation automatically produced by Atelier B (refer to section #7).

command-line interface

The user interface is started up, you already have opened a project.
To obtain an operation call graph, type the following command:
`op_call_graph`.
or :
`ocg`.

This command include three parameters:

1. the name of a component or the "`*`" value to have a graph on all the project.
2. the name of an operation, the `INITIALISATION` key word, or the "`*`" value to have the graphs of all the operations.
3. `1` if you wish to extend the graph to the refinements and to the operation abstractions, otherwise, `0`.

Example : To display the operation call graph of operation `op_name` of component `comp_name` and all its refinements and abstraction, type the following command :

```
op_call_graph comp_name op_name 1
```

or :

```
ocg comp_name op_name 1
```

6.6 Cross References

Description

This function performs searches on the identifiers defined in the project components.

For each identifiers, the function displays:

- its type: variable, set, ...
- the place where it is typed,
- the names of the components and the clauses where it is defined.
- the names of the components and the clauses where it is used.
- the names of the components and the clauses where it is modified (for variables only).

Identifiers are sorted by type.

The user can request this information:

- for a specific identifier,
- for all of the identifiers defined in a component,
- for all of the identifiers defined in the project: the user will obtain a dictionary of the terms used in the project.
  To distinguish the specifications and implemantaions of local operations, the names of the implemantations of local operations are preceded by `refinement_of_`.

Example:

```
-----------------------------------------------------------------------
VARIABLES
end_delay
        concrete variable
          Definition of "end_delay" in B_Delais.mch (CONCRETE_VARIABLES)
          Use of "end_delay" in B_Delay.mch (INVARIANT)
          Use of "end_delay" in B_Delay.mch (INVARIANT)
          Modification of "end_delay" in B_Delay.mch (INITIALISATION)
          Modification of "end_delay" in B_Delay.mch (b_init_delay)
          Modification of "end_delay" in B_Delay.mch (b_stopper_delay)
          Modification of "end_delay" in B_Delay.mch (b_delay_elapsed)

.....
-----------------------------------------------------------------------
OPERATIONS
b_delay_elapsed
        operation name
          Definition of "b_delay_elapsed" in B_Delay.mch
.....
-----------------------------------------------------------------------
OPERATION PARAMETERS
end_del
        operation output parameter
          Definition of "end_del" in B_Delay.mch (b_delay_elapsed)
          Modification of "end_del" in B_Delay.mch (b_delay_elapsed)
```

This information can be included in the project documentation automatically generated by Atelier B.

Warning: Calling this function generates a semantic analysis of the concerned components.

Command-line interface

The user interface is started up, you already have opened a project.
To obtain cross references, type the following command:
`get_project_xref`
or
`gpx`
This command includes one or two parameters:

1. The filter on identifiers:
    - `0` for all identifiers of the component specified as second parameter,
    - `1` for on identifier of the current project specified as second parameter,
    - `2` for all identifiers of the project,
2. The component for filter 0 or identifier for filter 1.

Examples:

To obtain cross references for all of the identifiers defined in the component `comp`, type the following command:
`get_project_xref 0 comp`

To obtain cross references on an identifier `ident`, type the following command:
`get_project_xref 1 ident`

To obtain cross references for all the identifiers defined in the project, type the following command:
`get_project_xref 2`

6.7 Extracting Metrics

Description

This function extracts metrics from an implementation or from all the implementations of the project.

These metrics are used to:

- measure the complexity of an implementation or of the analysed project,
- check that the implementation or the analysed project complies with programming rules.
- measure the minimum memory size required for implementing the data layouts used in the B sources.

The metrics extracted are exctracted according to the data present in a configuration file. This configuration file contains the following information:

- The reference value for each metric.
- The list of metrics to display in the project report; the project report is a table that contains metrics for all the project implementations.
- The list of metrics to display in the implementation report; the implementation report is a table that contains metrics for a specified implementation.

- The list of metrics to display in the operation report; the operation report is a table that contains metrics for an operation in a specified implementation.

The user can use the configuration files provided by Atelier B or create his own configuration files.

The configuration files supplied with Atelier B are located in the `AB/press/lib/LC` directory. The files have a `.cvl` extension.

Example of a project report:

```
+--------------------+---------+---------+---------+---------+---------+---------+
| Name               |     (1) |     (2) |     (3) |     (4) |     (5) |     (6) |
+--------------------+---------+---------+---------+---------+---------+---------+
| Reference Values   |     500 |     100 |      10 |       3 |       2 |     100 |
+--------------------+---------+---------+---------+---------+---------+---------+
| Distributor_imp    |      14 |      12 |       1 |       2 |       1 |       5 |
+--------------------+---------+---------+---------+---------+---------+---------+
| Screen_imp         |      13 |       6 |       1 |     !4! |       1 |       1 |
+--------------------+---------+---------+---------+---------+---------+---------+

(1)=NB_INST_OPER
(2)=NB_INST_SEQ
(3)=NB_CTRL_SEQ
(4)=NB_CTRL_IMB
(5)=NB_WHILE_IMB
(6)=LG_CONDITION
```

The first line shows the names of the metrics that are repeated after the table.
The second line remains the reference value for each metric.
The next lines show the values of the metrics for each project implementation.
If a metric value exceeds the reference, it is enclosed between `!<value>!` (example `!4!`).

Example of an implementation report:

```
METRICS FOR IMPLEMENTATION : Ecran_imp

+----------------------+------------+------------+------------+------------+
| Title                |   Value    |    Ref     |   % > ref  |    CR      |
+----------------------+------------+------------+------------+------------+
| NB_INST_OPER         |         13 |        500 |    --      |    OK      |
+----------------------+------------+------------+------------+------------+
| NB_INST_SEQ          |          6 |        100 |    --      |    OK      |
+----------------------+------------+------------+------------+------------+
| NB_CTRL_SEQ          |          1 |         10 |    --      |    OK      |
+----------------------+------------+------------+------------+------------+
| NB_CTRL_IMB          |          4 |          3 |    25      |    KO      |
+----------------------+------------+------------+------------+------------+
| NB_WHILE_IMB         |          1 |          2 |    --      |    OK      |
```

```
+----------------------+------------+------------+------------+------------+
| LG_CONDITION         |          1 |        100 |          0 |    OK      |
+----------------------+------------+------------+------------+------------+
```

The first column shows the names of the metrics.
The second column shows the maximum value of each metric for all implementation operations.
The third column shows the reference value for each metric.
The fourth column shows the excess percentage according to the reference value.
The last column shows *OK* if the value is below the reference value, *KO* if not.

Example of an operation report:

```
METRICS FOR OPERATION : message_controler_code

+----------------------+------------+------------+------------+------------+
| Title                |   Value    |    Ref     |   % > ref  |    CR      |
+----------------------+------------+------------+------------+------------+
| NB_INST_OPER         |          6 |        500 |    --      |    OK      |
+----------------------+------------+------------+------------+------------+
| NB_INST_SEQ          |          4 |        100 |    --      |    OK      |
+----------------------+------------+------------+------------+------------+
| NB_CTRL_IMB          |          4 |          3 |    25      |    KO      |
+----------------------+------------+------------+------------+------------+
| NB_CTRL_SEQ          |          1 |         10 |    --      |    OK      |
+----------------------+------------+------------+------------+------------+
| NB_WHILE_IMB        |          0 |          2 |    --      |    OK      |
+----------------------+------------+------------+------------+------------+
| LG_CONDITION         |          1 |        100 |          0 |    OK      |
+----------------------+------------+------------+------------+------------+
```

The second column shows the value of each metric for the analised operation. The meaning of the other table columns is the same as in the implementation report.

The following table shows the list of the available metrics:

Metric Code	Calculation performed
NB_SEE_MACH	number of seen machines
NB_IMPORT_MACH	number of imported machines
NB_MACH_EXTEND	number of extended machines
NB_ENUM_SET	number of enumerated sets
NB_ABSTR_SET	number of abstract sets
NB_ENUM_ITEM	maximum number of elements in a enumerated set
NB_MACH_PAR	number of machine formal parameters
TOT_NB_ENUM	total number of enumerated elements
NB_CONC_VAR	number of concrete variables
NB_OPER	number of operations
NB_CONST	number of concrete constants
NB_LOC_VAR	maximum number of local variables per operation
NB_INPUT_PAR	number of input parameters per operation
NB_OUTPUT_PAR	number of output parameters per operation
NB_INST_OPER	total number of instructions in an operation
NB_NEST_CTRL	maximum number of control statements nested in an operation
NB_SEQ_CTRL	maximum number of control statements in sequence in an operation
NB_NEST_WHILE	number of nested whiles
NB_SEQ_INST	number of instructions in sequence in an operation
NB_VAR_IN	number of VAR IN in an operation
LG_PREFIX	maximum size of rename prefixes
LG_PAR_MACH	maximum size of a machine parameter
LG_IMP	implementation name size
LG_INPUT_PAR	maximum size of an operation input parameter
LG_OUTPUT_PAR	maximum size of an operation output parameter
LG_SET	maximum size of a set identifier
LG_ITEM_SET	maximum size of a set element identifier
LG_CST	maximum size of a visible constant identifier
LG_CONC_VAR	maximum size of a visible variable identifier
LG_LITERAL	maximum size of a literal character string in an operation
LG_LOC_VAR	maximum size of a local variable identifier
LG_OPER	maximum size of an operation identifier
LG_OPER_MACH	maximum size of an operation identifier + machine identifier
LG_CONDITION	maximum size of a condition expression for an operation (number of operators)
SZ_ARRAY	memory space taken by the arrays
NB_OP_PROMUE	number of operations promoted and extended
SZ_CONC_VAR	memory space taken by the visible variables other than arrays
SZ_CST	memory space taken by visible constants other than arrays

Graphical user interface

This function is not available from the GUI.

Command-line interface

The user interface is started up, you already have opened a project.
To calculate the metrics on all the implementations of a project, type the following command:

`lchecker_project config_path output`
or
`lcp path_config output`

The first parameter is the complete path of the configuration file to use, for example `<rep_Atelierb>AB/press/lib/LC/CONFIG_clause.cvl`.
The second parameter is the output format; the values `0,1,4,6 and 8` correspond respectively to a LaTeX, Interleaf, ASCII, FrameMaker or Word displaying.
The values `2,3,5 and 7` correspond respectively to a LaTeX, Interleaf, ASCII, FrameMaker or Word print out.

To calculate metrics on a project implementation, type the following command:
`lchecker_mach name_imp config_path output`
or
`lcm imp_name config_path output`

The first parameter is the implementation name. The two other one have the same meaning that for `lcp`.

7 B projects Documentation

7.1 Description

Theses functions are available only if some external tools are installed on the workstation, like LaTeX, a PDF viewer and Microsft Word / OpenOffice.

The Atelier B provides some commands allowing to create automatically complete documents including the following information :

- B source files,
- user rule files (`.pmm`),
- status tables for the project and each component,
- different graphs,
- cross references.

In these documents:

- the B language symbols are displayed using math fonts.
- the B language key words are displayed in bold face characters.

The two commands offered by Atelier B are:

- Displaying a B source only.
- Create a complete document that may contain all the above information in the order chosen by the user.

7.2 Displaying a B Source

Description

This function applies itself on any component of a B project. The B source file (specification or refinement) is converted in an understable format of the selected word processor.

When converting the B source file, the user can choose two types of presentations, he can:

- keep his original presentation,

- use the presentation supplied by Atelier B.

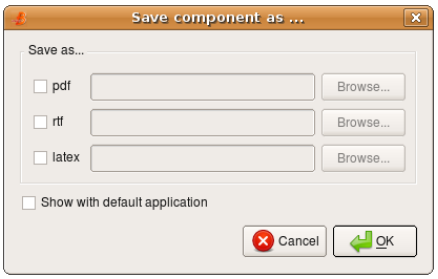
Normal comments are deleted from the B source file, only the comments enclosed between "`/*?`" and "`?*/`" character sequences are retained.

Graphical user interface

The GUI is already launched, and a B project is opened.

1. Select the desired component into the components list,
2. Trigger the *"Component -> Save as ..."* menu.

The following dialog is then displayed :



Select the export options, then finally click on the *"Ok"* button.

Command-line interface

The user interface is started up, you already have opened a project.
To display-convert a B source into a word processor format, use one of the following commands:

- `show_doc_latex` or `sdl`: This function converts the B source into LaTeX format.
- `print_doc_latex` or `pdl`: This function converts the B source into LaTeX format, then sends the produced file automatically sent to the selected printer. To change the selected printer, use the `set_print_params`command.
- `create_doc_rtf` or `cdr`: This function converts the B source into *Word* format. The name of the file produced is displayed in the start-up window. You should then edit the file using *Word*, or any *RTF-compliant* editor.

The first parameter for these functions is the component name.
The second parameter is the type of required presentation. To retain the original presentation, specify the value `PLAIN`. Use the `NORM` value for a standardised presentation.

Usable parameters

*ATB*OPT_ TOOLS_<SYSTEM>*Latex_ Binary_ Directory*
Positioned at Atelier B installation.
Directory where to find the Latex binaries.

8 Atelier B Parameters Customization

8.1 Introduction

This section describe how the user can tweak the Atelier B, in order to modify:

- external tools used by Atelier B (editor, HTML browser, etc ...),
- memory amount used by Atelier B,
- parameters of Atelier B tools,
- printing script used for documents generated by Atelier B.

8.2 Presentation of the customization system

There are three ways for the user to customize Atelier B :

- create a resource file `$HOME/.AtelierB` which customizes all Atelier B for the current user ; please not that we do not recommend to use this file, because it may lead to conflicts with the other configuration files,
- create an `AtelierB` file in the PDB directory of a project. This file customizes `Atelier B` when the user opens this project,
- create any file that the user has to specify explicitly when its use is requested.

When the same resource is specified in several of those files, the priority order is the following (from highest to lowest) :

- the explicit file,
- the file associated to the project,
- the file associated to the user.

When two files are given explicitly, the second one takes precedence.

When a resource is described in none of those files, a default value is taken, which is present in the Atelier B general resource file.

Atelier B parameters are, by default, defined in file `AB/AtelierB`. This file contains parameters shared by all atelierb users. It is also possible to define specific parameters for a user or a project (refer to *Atelier B User Manual*)

This file has the following format : Lines beginning with an exclamation mark `!` or a hash sign `#` are comments; other lines contain the name of a resource followed by its value.

The file contains the following parts :

- Graphical resources :
  `benv*<resource_name>`
- Localisation of internal tools :
  `ATB*ATB*<resource_name>`
- Localisation of optional tools :
  `ATB*OPT_TOOLS_<system>*<resource_name>`
  where `<system>` is `LINUX` for a Linux machine, `HP10` for HP-UX 10.20, `SUN5_6` for Solaris 6 or greater.
- Parameters for internal tools :
  `ATB*<internal_tool>*<resource_name>`.

Examples :

`ATB*POG*Generate_Obvious_PO: FALSE`
When the value of this resource is FALSE, Atelier B does not save obvious proof obligations.

To modify a resource for all Atelier B users, the procedure is :

1. Log-in as **atelierb** (under UNIX-compatible systems only),
2. Go to the Atelier B installation directory,
3. Edit the resource file `AtelierB`,
4. Change the line corresponding to the resource with the new value,
5. Save your changes.

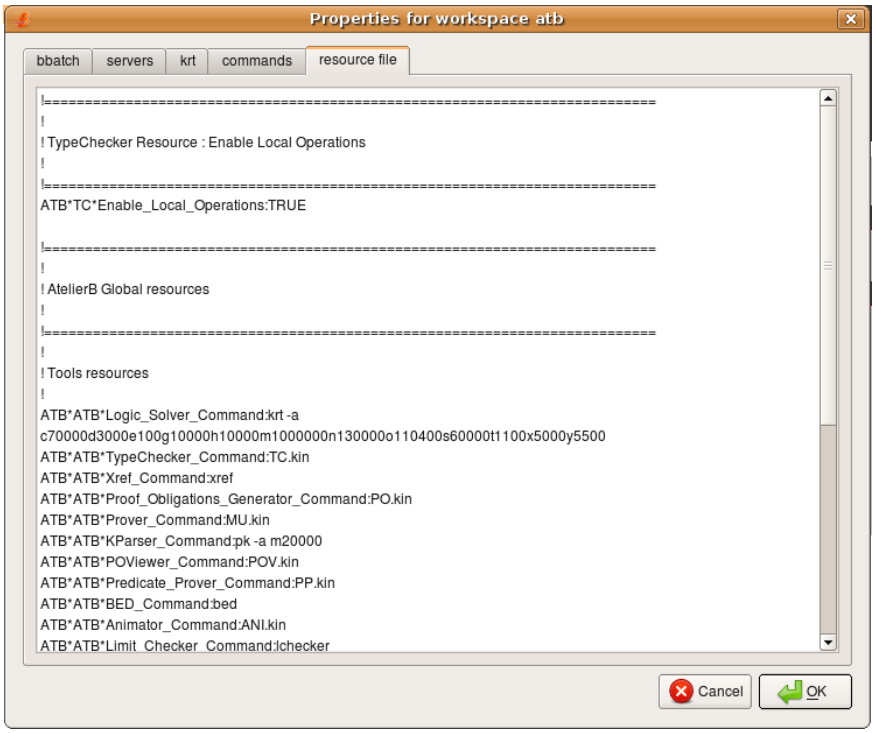
8.3 Creating a resource file

Description

This function allows to edit the resource file. If the file does not exist, it is created and initialized from a model. The model contains the list of all resources. Displaying a resource can be made by uncommenting the line and updating the resource value.

Graphical user interface

In order to edit a resource file from a workspace, click on the target workspace, then use the *"Workspace -> Properties"* menu, and click on the *"Resource file"* tab. An overview of the dialog is given below :



To edit the project resource file, open the given project, trigger the *"Project -> Properties"* menu, then click on the *"resource file"* tab. A similar window as seen previously is then displayed.

8.4 Display resource values and AtelierB version

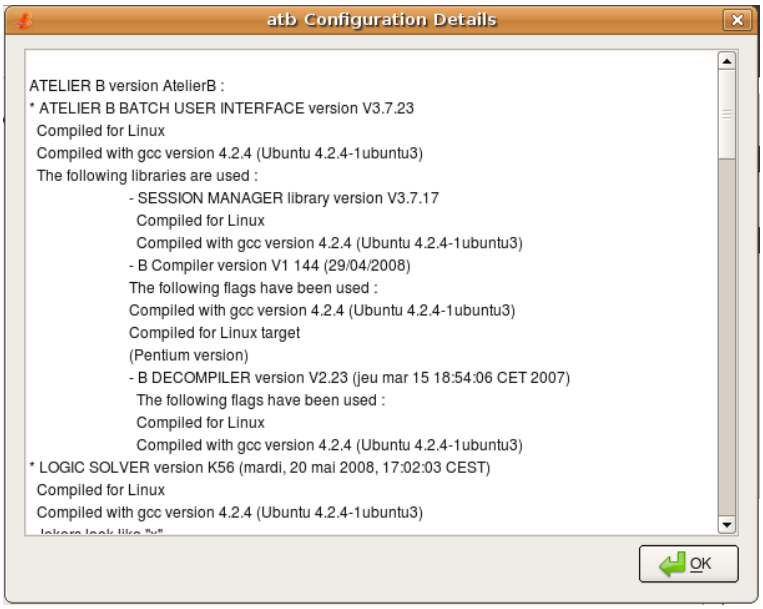
Description

The version display command shows :

- the global version of Atelier B,
- the version of all Atelier B tools,
- the current values of resources.

Graphical user interface

To display resource values and Atelier B version, just go on the workspace's properties, and on the first tab (*"bbatch"*), there is a *"Configuration details"* button. After having clicked on it, a window similar to the following screenshot :



8.5 Modifying the external tools configuration

Presentation

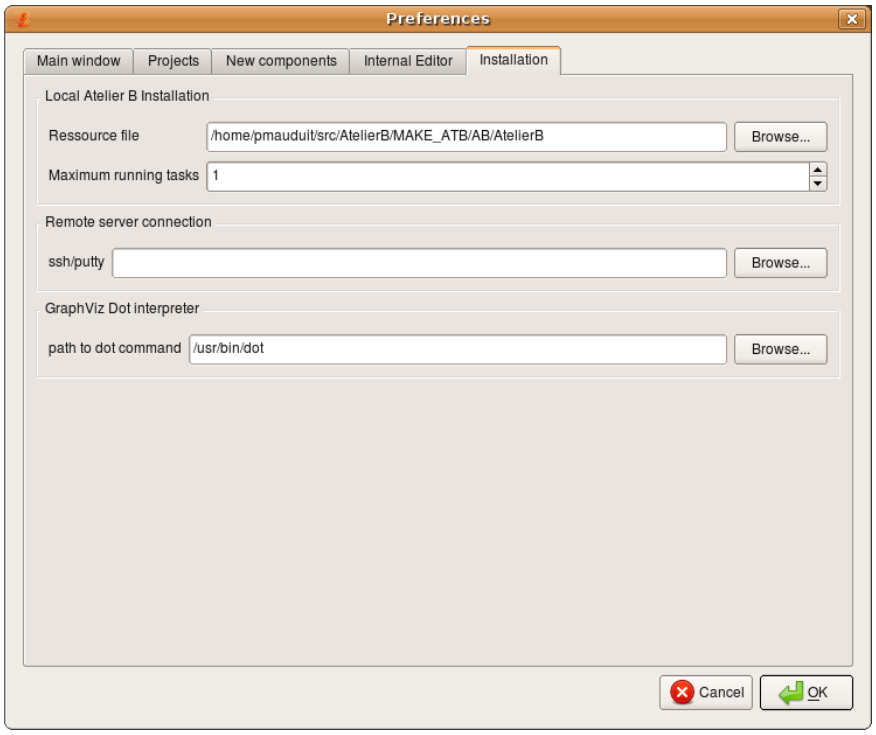
By default, Atelier B uses the external tools configured during installation.

To modify one of these tools for all Atelier B users, you have to modify the value of its resource :

Tool	Resource
Text editor	ATB*OPT_TOOLS_<SYSTEM>*Editor_path
LaTeX	ATB*OPT_TOOLS_<SYSTEM>*Latex_Binary_Directory

Graphical user interface

It is possible to configure some external tools directly from the GUI, independently of the resources explained previously. To do so, trigger the *"Atelier B -> Preferences"* menu, then click on the *"Installation"* tab. The following window is displayed :



From this interface, you can reconfigure the SSH and DOT paths.

8.6 Modifying Memory Allocation of the Logic Solver

Modifications of the Logic Solver parameters are required in the following cases:

- Your machine does not have enough memory to execute Atelier B.
  Message type:
  ```
  Cannot launch the Logic Solver
  (check if there is enough memory)
  ```
- The complexity of your B project, forces you to increase the memory size.
  Message types:
  ```
  Compiler Memory Full
  SEQUENCE ID NUMBERS OVERFLOW
  SEQUENCE MEMORY OVERFLOW
  SEQUENCE _chn ID NUMBERS OVERFLOW
  SEQUENCE _chn MEMORY OVERFLOW
  stopping forward because ...
  OBJECTS OVERFLOW
  MAXIMUM NUMBER OF THEORIES xx REACHED
  ```

```
GOAL STACKS OVERFLOW
SYMBOLS OVERFLOW at <symb>
```

To set Logic Solver parameters for all Atelier B users, it is necessary to modify the value of the resource named `ATB*ATB*Logic_Solver_Command` in file `AB/AtelierB` (Refer to section #8.2).

Depending on the size you need, you have to replace the line
`ATB*ATB*Logic_Solver_Command: krt`
with one of the following ones:

- for a small Logic Solver :
  `ATB*ATB*Logic_Solver_Command: krt -a m700000e40`
- for a medium Logic Solver :
  `ATB*ATB*Logic_Solver_Command: krt -a m1000000e40`
- for a large Logic Solver :
  `ATB*ATB*Logic_Solver_Command: krt -a m4000000e40`
- for an extra-large Logic Solver :
  `ATB*ATB*Logic_Solver_Command: krt -a m6000000e40`

If a user wants a Logic Solver with another size (different from the size used by all Atelier B users), he must write its value for the resource `ATB*ATB*Logic_Solver_Command` in a file named `.AtelierB` in his home directory.

8.7 Modifying Memory allocation of the K parser

Modification of the K parser parameters must be performed if your B source files are very large or if these files contain a large number identifiers:

Messages like:

```
SEQUENCE ID NUMBERS OVERFLOW
SEQUENCE MEMORY OVERFLOW
SEQUENCE _chn ID NUMBERS OVERFLOW
SEQUENCE _chn MEMORY OVERFLOW
SYMBOLS OVERFLOW at <symb>
```

To modify the K parser parameter settings, please follow the same procedures as above, using the resource named `ATB*ATB*KParser_Command`.

8.8 Configuring the Printing Script

Description

By default, the printing script used by Atelier B is the script named
`atelierb_directory/AB/bbin/bprint`

To use another printing script:

1. create a new script file
2. modify the value of the resource `ATB*ATB*Print_Command` in the file `AB/AtelierB` with the full path of the new script file.

The Atelier B is shipped with a printing tool named `bprint`. This script is unavailable from the GUI, but could be used via the Command-line interface.

By default, the printing script used by the Atelier B is the the script *atelierb_ directory/AB/bbin/bprint*. In order to use another script, follow these instructions:

1. Create a new script,
2. Modify the value of the resource `ATB*ATB*Print\_Command` in the resource file `AB/AtelierB` by giving the full path of the new script.

Appendix

Limitations of Project Documentation Tools

The outputs in *Word* (`.rtf`) formats are limited. This is due to the abilities of the programs and their formats. This functionality should be used with care.

For the Word output format, the limitations are the following :

1. The logo is not included in the generated document.
2. The table of contents is not generated.
3. Be careful when including Postscript files:
    - no check is performed to ensure that the file is present on generation;
    - the file must be present on the disk;
    - Word must recognize Postscript files;
    - The printer must be able to interpret the Postscript.
4. The files are intended for PCs and may not be used directly on Macintosh computers.

Files created by the Atelier B

The table below describes all the files created by Atelier B:

File	Location	Contents
`project_name.desc`	Atelier data base	Project descriptor (directories, manager, users, libraries)
`project_name.db`	Project data base	Project components (name, localization, owner)
`.usedby_*`	Project data base	Marker indicating that the project is opened by a user
`.project`	Project data base, translation directory	Marker indicating that the directory is occupied by a project
`.lib`	Project data base	Directory of library project PDBs
`*.lock`	Project data base	Markers used to ensure mutual exclusion between users of the same project
`deB*,versB*`	Project data base	FIFOs for communication between the user interface and the Logic Solver
`src/*.*`	Project data base	B source files with expanded definitions. There is one file per component even if
`expand_src/*.*`	Project data base	B source files, one per component
`*.nf`	Project data base	normalized form of component
`*.tse`	Project data base	Extended table of symbols generated by B0Checker
`*.po`	Project data base	Component proof obligations
`*.opo`	Project data base	component obvious proof obligations
`*.pmi`	Project data base	Saved component interactive proof
`*.pmm`	Project data base	Rules defined by the user for each component
`project_name.gdl`	Project data base	Dependency graph
`project_name.stg`	Project data base	"Project Status" table in SGML format.
`*.stg`	Project data base	"Component Status" table in SGML format.
`*.tex`	Project data base	Files generated by the documentation tools for translation into LaTeX.
`*.rtf`	Project data base	Files generated by the documentation tools for translation into Word.
`*.dvi`	Project data base	Files generated by LaTeX for display or print-out.
`*.ps`	Project data base	LaTeX files converted into PostScript format for printing-out.
`*.bod, *.str, *.blf`	Translation directory sub-directory `ada`	Object files generated by the ADA translator.
`*.ads,*.adb`	Translation directory sub-directory `ada`	Files generated by the ADA translator after link edition.

continued on next page

continued from previous page

File	Location	Contents
`makefile`	Translation directory sub-directory `ada`	Directives for ADA compiler.
`*.bod, *.str *.blf`	Translation directory sub-directory `hia`	Object files generated by the HIA translator.
`*.hia,*.h`	Translation directory sub-directory `hia`	Files generated by the HIA translator after link edition.
`makefile`	Translation directory sub-directory `hia`	Directives for HIA compiler.
`*.bdy, *.spe *.blf`	Translation directory sub-directory `cpp`	Object files generated by the C++ translator.
`*.cpp,*.h`	Translation directory sub-directory `cpp`	Files generated by the C++ translator after link edition.
`makefile`	Translation directory sub-directory `cpp`	Directives for C++ compiler.
`*.bdy, *.spe *.blf`	Translation directory sub-directory `c`	Object files generated by the C translator.
`*.cpp,*.h`	Translation directory sub-directory `c`	Files generated by the C translator after link edition.
`makefile`	Translation directory sub-directory `c`	Directives for C compiler.